Simple Algorithms for Hot-Potato Routing

Thesis for the M.Sc. Degree

by

Krauthgamer Robert

Under the Supervision of
Dr. Uriel Feige
Department of Applied Mathematics
The Weizmann Institute of Science

Submitted to the Feinberg Graduate School of
the Weizmann Institute of Science
Rehovot 76100, Israel

October 24, 1996

Acknowledgments

First, let me express my deep gratitude to my advisor, Dr. Uri Feige, for providing me with his remarkable guidance, insights and helpful suggestions, for all the fruitful discussions we held, and for always being available.

In addition, I would like to thank Nati Linial and Ori Sasson for intriguing discussions and helpful comments.

Many thanks also to my friends from the Weizmann Institute of Science for pleasant company, interesting discussions, and valuable advice. In particular, I enjoyed the exciting company of Tal, Son, Ovadya, Kobbi, and Amoss. Thanks, guys.

Finally, I would like to thank my girlfriend Yifat, my family and friends for their love and for their belief in me. Your love gives me confidence in my way.

Special thanks to my parents Arnon and Rita for their endless support and encouragement. Mum and dad, you deserve all the credit for motivating me since my childhood, and leading me now to pursue scientific research.

Abstract

Hot-potato routing is a particular form of routing in a synchronous network of processors, which makes no use of buffers at intermediate nodes. Packets must keep moving in the network (possibly deflected to "bad" directions), giving rise to the term "hot-potato". Simple hot-potato algorithms are interesting for both practical and theoretical reasons.

We analyze the worst case performance of some simple hot-potato routing algorithms. For example, we show that any "minimum advance" algorithm cannot livelock on a tree network, and present a deterministic algorithm for general graphs, inspired by random walks.

One of our main topics studies an algorithm for the mesh based on selecting a small number of Hamiltonian paths, such that vertices that are close together on the mesh are also close together on at least one of these paths. Based on these families of Hamiltonian paths, routing between vertices is achieved in time that depends only on the distance between these vertices, regardless of the size of the whole mesh.

This framework of mapping meshes to several Hamiltonian paths may be of independent interest, and indeed we demonstrate its relevance to the construction of hash functions.

Contents

Chapter 1

Introduction

1.1 Background

This work studies packet routing in synchronous network of processors, in which any communication link can carry at most one packet at each time step. We consider a form of packet routing known as *hot-potato routing* or *deflection routing* [1, 12, 13, 15, 21, 27, 28]. The important characteristic of this form is that it requires no buffer space for storing delayed packets. Each packet must leave the processor at the step following its arrival, unless it has arrived to its destination. Packets arriving to a processor from its neighbors have to be redirected to distinct outgoing links. That is, packets keep moving, giving rise to the term "hot-potato". This may cause some packets to be "deflected" away from their preferred direction. In particular, some packets may temporarily move further away from their destinations. This differs from the traditional store-and-forward routing, where a packet can be temporarily stored at a processor, and forwarded along the desired link once it becomes available.

In general, deflections cause a packet to traverse more edges on its way to its destination, compared with store-and-forward routing. This is an added burden on the network resources (communication links). On the other hand, deflection eliminates the need for storing packets at intermediate processors (other than the source and destination). Storage operations may be undesirable, as they require processors to allocate memory for packets not addressed to them, potentially consuming resources such as time and energy. If a packet is not going to be stored, there is no need to read in the whole packet, and the packet header suffices for the routing decisions.

We find the study of natural and simple hot-potato routing algorithms interesting for its own right, regardless of potential applications. Nevertheless, this research may have practical significance, as hot-potato routing trades off memory resources to communication links. The desirability of such tradeoff depends on the available resources and technology. Parallel machines such as the Connection Machine [16] and the HEP multiprocessor [26], as well as high-speed communication networks [21] use deflection routing in various forms.

Deflection routing is also highly desirable in the domain of optical networks [1, 13, 27, 28]. In such networks, the storing of packets require conversions between optical form and electronic one. Currently, these conversions are much slower than optical transmission rates, and it pays off to deflect a blocked message, even at the cost of longer routes.

Intermediate processors in hot-potato routing have no queues of waiting packets, enabling shorter cycle times. In order to capitalize on this feature, it is important that the routing choices a node makes are simple and easy to implement.

1.2 Preliminary Considerations

A routing algorithm for a network is a collection of functions, one function for each node of the network. The arguments of the function are the header information of the packets that enter the node at a particular time step, and the internal state information of the node. The output of the function is an assignment of the packets to distinct outgoing edges, except for those packets destined to the current node.

Given a network and a routing algorithm for that network, we would like the algorithm to perform well on routing problems that occur in practice. We present here some typical parameters to consider.

Batch vs. dynamic routing: Batch routing problems are of a "one shot" nature, where several packets are injected into the network simultaneously, and after that, no new packet is injected into the network (until all packets from the first batch reach their destination). In dynamic routing problems, packets are continuously being injected into the network. Dynamic routing scenarios typically allow for a source node to store its own packets, since in certain time steps it might be impossible to inject a packet into the network (if all outgoing edges from the source node are used by other packets).

Worst case vs. average case: We may measure a routing algorithm according to its performance on a worst case routing problem. Alternatively, a probability distribution can be defined over the possible routing problems, and then we can measure average case performance or performance with high probability.

Evacuation vs. bound per packet: For batch routing, it is typical to consider the *evacuation time* - the time until all packets reach their destinations. Dynamic routing must measure the *delivery time* - time required for a typical individual packet to reach its destination.

Load: The performance of algorithms may vary with the load (number of packets) in the network. For a lightly loaded network, one would expect packets to reach their destinations quickly, whereas for a heavily loaded network, larger delays are to be expected.

Class of Networks: It is usually easier to devise an algorithm which is restricted to a certain class of networks, such as trees, meshes, hypercubes etc. Commonly, the algorithm is tailored to the network layout, in order to ease performance analysis. Algorithms for general networks usually cannot exploit such specific properties.

Capacity: Communication links can carry only a limited quantity of packets at each time step, called the *capacity* of the link. Commonly, the capacity of each link is limited to one packet in each direction, but in some cases we may assume links of larger capacity.

In this work, we restrict ourselves to a certain class of deflection networks, that correspond to the class of undirected graphs. We do not allow self-loops nor parallel edges, and each undirected edge (link) represents a pair of anti-parallel edges.

We consider only routing algorithms that do not alter the packets (and, in particular, their headers), after the packets are injected to the network.

We are concerned only with worst case bounds. We do not consider any kind of average case nor randomness in the routing algorithm or in the routing problem. Some variants of hot-potato routing allow the annihilation of a packet in "severe" situations, which happen with very low probability. We do not allow this feature.

In general, we will concentrate on batch routing.

We introduce some basic notation which will be used throughout this work:
$\boldsymbol{n}$ (or $\boldsymbol{N}$) - number of nodes in the network. For meshes and tori, $\boldsymbol{n}$ will denote a side of the mesh (e.g. the $\boldsymbol{n} \times \boldsymbol{n}$ mesh).
$\boldsymbol{k}$ - number of packets in the network.
$\boldsymbol{d(u, v)}$ - the distance (length of shortest path) between nodes $\boldsymbol{u}$ and $\boldsymbol{v}$.
$\boldsymbol{D}$ - diameter of network.
$\boldsymbol{d}_p$ - the distance between source and destination of packet $\boldsymbol{p}$.
$\boldsymbol{t}_p$ - the number of time steps until packet $\boldsymbol{p}$ reaches its destination.

The least requirement from a routing algorithm is a guarantee that all packets will reach their destination at some time. It would be unfortunate if the algorithm enters *livelock* - packets keep cycling in the network and never reach their destination. Routing algorithm might perform well on some networks, and livelock on others.

When measuring the performance of an algorithm, we usually seek an explicit bound on the evacuation time or on the delivery time (t_p). The advantage of bounding the delivery time (of a packet) over the evacuation time (of the network) is the option to refer to parameters which are specific to this packet. For example, delivery time bound can be expressed in terms of d_p. The corresponding evacuation time would then use D as an upper bound for d_p.

In general, we would like a worst case bound which depends on the network load (k) and on the distances between source and destination of packets (d_p). Even a tight worst case bound is not necessarily practical, as practical implementations take into account also the average case. It is also important to figure out the problem nature. For example, when $k >> d_p$, one would like to bound performance as a function of d_p alone.

In many cases a lower bound on the performance is also interesting, and not only for disqualifying unsuccessful algorithms. When related to a whole class of algorithms, a lower bound points out a weakness of this whole class, and might lead to a corresponding refinement of the class. When related to a specific algorithm, the lower bound is typically used to evaluate the quality of corresponding upper bounds.

1.3 Classification of Algorithms

We are interested in natural algorithms which are simple to implement. A coarse separation of the routing algorithms into classes includes shortest paths algorithms and structured algorithms (we will shortly define these terms as we discuss them).

In *shortest path* algorithms, a packet which does not meet any other packets on its way, will be routed along a shortest path towards its destination. Following [11], we further refine this class[1] by restricting the behavior in the case that several packets meet in a node. The restrictions below (additional restriction further limits the class of algorithms) apply at any time step, for any nonempty node.

- **Minimum Advance:** At least one packet advances towards its destination.
- **Weakly Stable:** If a packet is deflected, than there is no free outgoing edge that would lead it closer to its destination.
- **Stable**[2]**:** There is no possibility of changing the edges assigned to some of the packets such that all those packets strictly gain from the change (get an edge that leads them closer to their destination).
- **Maximum Advance:** The maximum possible number of packets advance.

These restrictions by themselves do not guarantee good performance. Hajek (Fig. 1 in [15]) demonstrated a livelock situation on the 4-cycle (and hence also on the mesh) for a stable algorithm. Feige [11] exhibited a livelock situation for a maximum advance algorithm, on a permutation problem on certain networks (including the torus).

Hence, the maximum advance principle is not sufficient by itself to guarantee termination. This is due to the lack of a good *contention resolution* - a method of assigning priorities to contesting packets. Two examples of priority rules which avoid livelock are:

1. Priority is given to packets that are closest to destination.

[1]Similar classes, but using different terminology, appear in [6, 5].

[2]This corresponds to a stable solution in terms of game theory.

2. The packets have fixed priority levels, independently of their location in the network.

It is easy to show that any minimum advance routing algorithm obeying one of the above two priority schemes routes a batch of k packets in at most $k \cdot D$ steps. The maximum advance livelock mentioned above is based on furthest first priority scheme.

Another class of algorithms is the *structured* algorithms, in which packets follow some predetermined policy. The policy is well chosen, so that packets travel in a pre-specified global order. These routes are not the shortest possible, but their analysis is usually simpler because of their rigid pattern. Structured algorithms are not necessarily even minimum advance, as they follow the fixed, pre-specified route even if the destination can be reached by a shorter path.

A subclass of the structured routing algorithms are *collision-free* algorithms, where packets routes are carefully planned to avoid contention. In collision-free algorithms, the route of each packet is set in advance, independently of the other packets. Therefore, the delivery time t_p is independent of the load k. A naive example for such algorithm is routing along a Hamiltonian path (or cycle). Assume the network contains a Hamiltonian path, which visits each of its n nodes. If any node in the batch problem is the source of at most one packet, the network is evacuated within n steps, regardless of the number of packets, k.

In this work (Chapter 4) we also consider the scheme of *potential guided algorithms*, where each packet has some potential value at each of the network nodes. The algorithm at each node assigns packets to outgoing edges such that the sum of potentials (at the next step) will be minimal. This definition alone does not guarantee good performance, and additional restrictions on the potential function are required.

Other structured algorithms, based on various schemes, are sometimes tailored to certain networks. For example, Feige and Raghavan[12] presented the *three bend algorithm* for two dimensional torus networks. In their algorithm, packets move basically along rows and columns, and the number of turns (change of direction) is restricted to 3. Another example is routing by (essentially) sorting, devised by Newman and Schuster [23], for mesh, torus and hypercube networks.

1.4 Summary of Results

Recall the concept of collision-free algorithms, where packet routes are pre-specified, so packets never contest on the same link. A straightforward collision-free algorithm is routing along a Hamiltonian path. Recall that a Hamiltonian path is one which visits every node in the graph exactly once. Assuming that each packet is the source of at most once packet, we get $t_p \leq n$. However, this is quite poor in many cases, e.g. where $n >> d_p$. This is the case on the two dimensional mesh (or torus), where the diameter is $O(\sqrt{n})$.

We address the problem of designing a collision-free routing algorithm in which the time required to deliver a packet is independent of the network size and load, i.e. the bound for t_p depends solely on d_p. We deal with mesh and torus networks, focusing on the two dimensional case.

We suggest to use a family of Hamiltonian paths instead of a single path. When injecting a packet, one of the Hamiltonian paths is chosen, and the packet travels along this Hamiltonian path with no interference from other packets, until reaching its destination. We require that for each (possible) packet p at least one of the Hamiltonian paths will deliver p to its destination within time which depends only on d_p, say $g(d_p)$ steps. An elementary argument shows that using only one path cannot satisfy these requirements.

In Chapter 2 we study the underlying model. It is a family $\mathcal{F}$ of Hamiltonian paths (or cycles) on the mesh (or torus), such that any two vertices which are d-far from each other on the mesh, are connected by (at least) one of the paths in $\mathcal{F}$ within some *expansion* $g(d)$. The performance measurements of this model are the cardinality of the family $\mathcal{F}$, and the order of magnitude of the function g. We investigate the asymptotic behavior of these measures, i.e. when $n \to \infty$.

The underlying model is analogous to stereoscopic[3] vision, where the combination of several two dimensional images give the appearance of the original solid form in three dimensions, particularly its depth (distance) information. Similarly, in our model a collection of Hamiltonian paths jointly characterize the distance information of the original mesh.

We thus refer to this $\mathcal{F}$ as a *stereoscopic family of permutations*, (Hamiltonian paths are generalized to arbitrary permutations).

We construct a stereoscopic family of permutations for the two dimensional mesh (or torus). This family consists of 3 Hamiltonian paths (or cycles), and achieves quadratic expansion, $g(d_p) = O({d_p}^2)$. We show that two paths cannot satisfy such requirements, so this construction is optimal w.r.t. $|\mathcal{F}|$. We also show that for any family $\mathcal{F}$ of constant cardinality, $g(d_p) = \Omega({d_p}^2)$ by a simple counting argument (regarding the sizes of the corresponding neighborhoods), so up to constant factors, our expansion g is optimal.

We conclude that routing by stereoscopic families of permutations in the two dimensional mesh achieves $t_p = \Theta({d_p}^2)$, with the requirement that the capacity of each link is 3 (to support the 3 Hamiltonian paths).

Stereoscopic families of permutations are applicable also in other areas, such as approximate queries in a dictionary and hashing noisy data. Linial and Sasson [20] present a hashing scheme which is *non-expansive* - close inputs are stored close to each other in memory. Their scheme hashes a one dimensional universe (interval) into another interval (in fact a small set of intervals). The framework of stereoscopic families of permutations allows us to generalize the one dimensional scheme to arbitrary dimensions.

[3]From Greek: stereo's = solid and skopein = to look at/view

For such applications, we study the case of higher dimensions and extend the two dimensional construction to $m+1$ Hamiltonian paths in the m-dimensional torus, with $g(d) = O(d^m)$ expansion. This family of Hamiltonian paths is then used to construct a stereoscopic family of permutations for the general case, mapping an m-dimensional universe to a q-dimensional one, with $m+1$ one-to-one functions and achieving $g(d) = O(d^{m/q})$ expansion.

We demonstrate how the general case construction can be used to upgrade a one-dimensional non-expansive hashing scheme, to inputs and memory of arbitrary (possibly different) dimensions. An m-dimensional universe is hashed to a q-dimensional one, simultaneously by $m+1$ one-to-one functions, each having its own copy of the whole dictionary. This scheme maps distance Δ in the m-dimensional universe to $O(\Delta^{m/q})$ in the q-dimensional storage. However, the memory size required is $m+1$ times larger than in the one-dimensional scheme, and each operation must be performed in each of these $m+1$ copies.

In Chapter 3 we show that any minimum advance algorithm on any tree network cannot livelock. Our proof holds also in the presence of an adversary, as long as the minimum advance principle is satisfied.

In chapter 4 we define the class of *potential guided algorithm*, where each packet has a potential function, i.e. a potential value at each node in the network. The potential function might depend on the packet's current location, the packet's destination and source nodes, etc. Each node locally routes incoming packets so as to minimize their potential, where as contentions are resolved through the potentials. That is, each node applies the assignment which leads to the minimum possible "sum of potentials" in the next step (after traveling their assigned edges).

We show that any potential guided algorithm on any tree network is stable. However, a potential guided algorithm is not necessarily maximum advance, even not on tree networks.

We suggest a specific potential function which is the expected hitting time of random walk from the packet's current location to its destination. Using this potential function, any network is evacuated within $2n^3 \log k$ steps. On tree networks and cycles we show that this algorithm is maximum advance and achieves $t_p \leq d_p + 2(k-1)$. Unfortunately, this result does not hold for mesh networks, since even a single packet does not use a shortest path.

1.5 Related Work

Baran [3] is widely credited with having first proposed hot-potato routing. Borodin and Hopcroft [7] proposed an algorithm for hot-potato routing on the hypercube. Although they did not give a complete analysis of its behavior, they observed that "experimentally the algorithm appears promising". Prager [25] showed that the Borodin-

Hopcroft algorithm stops in n steps on the 2^n-nodes hypercube for a special class of permutations.

Apparently, the first to consider worst case bounds for hot-potato routing was Hajek [15]. Some of his results which are relevant to our work, are demonstrating the possibility of a livelock, and the notion of priority to packets that are closer to their destination.

Part of our work concerns worst case bounds for general networks. Using the closest first priority rule devised by Hajek, it is clear that any minimum advance algorithm on any network terminates in $O(k \cdot D)$. The work of Hajek was continued by Brassil and Cruz [9]. In the context of general networks, Brassil and Cruz considered a minimum advance algorithms with fixed priorities and obtained the following result. Let W_p be the length of the shortest path that connects, in order of decreasing priority, the destinations of packets with priority up to the priority of packet p. Let k_p be the number of packets with priority higher than that of packet p. Then $t_p \leq D + 2k_p + W_p$.

An upper bound for general networks was also considered by Feige [11]. He proved that any maximum advance algorithm with fixed priorities achieves $t_p \leq d_p + 2^k - 2$, and showed that this result is tight, i.e. in some maximum advance cases (including fixed priority and closest first priority) $t_p = d_p + 2^k - 2$. As a corollary, he concluded that any maximum advance algorithm with fixed priority evacuates any network within $O(D + k\frac{D}{\log D})$. In the same paper, Feige also shows an algorithm for arbitrary networks that achieves $t_p \leq 2(R + k - 1)$, where R is the network radius. This algorithm is based on applying the maximum advance principle on a spanning tree, and is not even minimum advance on the original network.

Some other part of our work refers only to mesh and torus networks. This area of worst case bounds for routing on networks of specific types (meshes, tori and hypercubes) was also studied. Newman and Schuster [23] presented an algorithm that is based on sorting for permutation routing in the $n \times n$ two dimensional mesh. Their algorithm routes every permutation in $7n + o(n)$ steps. They apply the same idea to route permutation problems on the 2^n hypercube in $O(n^2)$ steps, and on the $n \times n$ torus in $4n + o(n)$ steps. Using the same method, Kaufmann, Lauer and Schroder [18] improved the result for the mesh. Other works [4, 6] present simpler algorithms for the $n \times n$ two dimensional mesh and torus networks, but their bounds are not as good as $O(n)$.

Borodin, Rabani and Schieber [8] show an algorithm that achieves $t_p \leq d_p + 2(k-1)$ on mesh networks of arbitrary dimension, if each node is the source of at most one packet. Feige [11] shows an algorithm for mesh and torus networks of arbitrary dimension that achieves $t_p \leq d_p + 2k$ in any problem.

Other works involving average case analysis (or random algorithms) on specific networks include the works of Feige and Raghavan [12] and of Kaklamanis, Krizanc and Rao [17].

In some routing problems $k >> d_p$, and one wants to bound t_p as a function of d_p

alone. Ben-Aroya and Schuster [5] designed an algorithm that routes any one-to-one problem in a two dimensional mesh in 7 steps, provided that for all packets p, $d_p \leq 3$. Feige [11] improved this to 5 steps, and presented a stable algorithm that guarantees $t_p \leq O(d_p)$ on the infinite line.

Part of our work is applicable also in the domain of hashing noisy data. In this context, our work can be used to extend the non-expansive hashing scheme devised by Linial and Sasson [20]. Related work in this context is described in section 2.8, where we discuss this topic.

Chapter 2

Stereoscopic Families of Permutations

2.1 Motivation

Unlike Store-and-Forward routing, packets in hot-potato routing might be deflected to undesired locations, deviating from their preferred course. The actual route traveled by each packet in a hot-potato algorithm depends on many other packets and their interactions. It is therefore hard not only to design efficient hot-potato algorithms, but also to analyze performance of hot-potato routing.

A useful approach for overcoming all these deflections, is simply avoiding contention between packets, in a collision-free algorithm. Recall that we defined a collision-free algorithm as one in which packets never contest on the same link. So the route of each packet in a collision-free algorithms is independent of other packets, and thus, known in advance.

A straightforward way to design a collision-free hot-potato algorithm is routing along a Hamiltonian path (or cycle). Recall that a Hamiltonian path (cycle) is one which visits every node in the graph (network) exactly once.

On the one hand, routing along Hamiltonian paths is elegant, simple to describe and analyze, and easy to implement (i.e. the routing table in every node is trivial). Assume each node gives priority to packets traveling in the network over new packets generated at the node. Then once packets are injected to the network, they travel to their destination without any interference. Another advantage of routing along a Hamiltonian path is in batch routing, where every node is the origin of at most one packet. In this case the time required to deliver a packet is independent of the network load, even if all packets are destined to the same node.

On the other hand, the performance guaranteed by routing along a Hamiltonian path is not very good, since packets travel almost the whole network, $\Omega(N)$, regardless of the situation. This is especially poor when the packet's destination is relatively close on the network but quite far on the Hamiltonian path. Delivery time of $\Omega(N)$ is also unsatisfactory when the network diameter is much smaller than N, as in the two-dimensional mesh, where the diameter is $O(\sqrt{N})$. In both cases one would expect much better delivery time since packets have the possibility to travel significantly shorter routes than $\Theta(N)$.

Of the several shortcomings of the Hamiltonian path routing scheme, our work addresses only one: design a collision-free scheme in which the routing time for each individual packet can be bounded by a function of the distance between its source and destination, independent of the size of the network. We deal with mesh and torus networks, concentrating on the two dimensional case. Another property we would like to preserve is the flexibility to extend the batch (one-shot) algorithm to dynamic routing, allowing new packets to be generated continuously.

We mentioned before that routing along a Hamiltonian path on the two-dimensional mesh results with poor delivery time w.r.t. the distance d_p. In fact, a simple argument (see section 2.3.3) shows there will always be a pair of neighbors on the mesh whose distance on the Hamiltonian path is $\Omega(\sqrt{N})$. Thus, the delivery time of a single packet cannot be bounded by a function of d_p alone (independent of n).

Our idea is to enhance the collision-free hot-potato algorithm to simultaneously use several Hamiltonian paths, and route each packet along the best Hamiltonian path. In this approach, we first need a small family $\mathcal{F}$ of Hamiltonian paths. Then, routing from some origin x to destination y is performed by traveling along the best (w.r.t. x and y) Hamiltonian path from $\mathcal{F}$ (the path is chosen only once, when the packet is injected). We require that for each (possible) packet p there will be at least one Hamiltonian path in $\mathcal{F}$ delivering p within some $g(d_p)$ steps, (for example $O({d_p}^2)$).

We suggest the following interpretation to routing along Hamiltonian paths. Consider a factory, whose production floor forms a mesh, and its workers are located at each of the mesh nodes. These workers interact with each other by sending and receiving packages (products, messages etc.), along (one-directional) moving strips.

A simple solution is to use one Hamiltonian cycle to cover all workers posts, within $\Theta(N)$ time. Each worker picks from the strip packages destined to him, and puts on the strip those he wants to send, as soon as he spots a vacant place. This resembles the moving strips used to pick up baggage in airports, where baggage is added to the moving strip on a vacancy basis, and people take their belongings when it reaches their position.

An alternative setting for the factory moving strips is building two (one-directional) moving strips for each path in the stereoscopic family $\mathcal{F}$. The overall investment sums up to $2|\mathcal{F}|$ moving strips, each located in a distinct elevation (height), and covering the whole factory floor. The workers algorithm remains quite the same. Each worker that wants to send a package chooses a moving strip, according to a fixed table of packages destinations. He then puts his package on the strip, as soon as he spots a vacant place on this strip. When the worker sees a package destined to him on any of the $2|\mathcal{F}|$ strips, he picks it up.

The advantage gained by using $2|\mathcal{F}|$ moving strips is that packages are delivered within $\Theta({d_p}^2)$ time. If the size of the floor N is very large and/or workers cooperation tend to favor short distances, the performance with $2|\mathcal{F}|$ strips is significantly better than with only one strip. Each worker has to be alert to the $2|\mathcal{F}|$ strips at all times. Hence, we want $\mathcal{F}$ to be of small cardinality as much as possible. Smaller $|\mathcal{F}|$ also require less resources (strips) to be put in the system.

Allowing several Hamiltonian paths complicates the simple collision-free hot-potato routing (see section 2.7 for details), but also puts the grounds to an independent theoretic model of preserving distances (metrics) when mapping a high dimensional mesh to a lower dimensional one (e.g. two dimensional mesh mapped to the line). Clearly, reducing the dimension is achieved on the expense of other resources, such as the number of Hamiltonian paths $|\mathcal{F}|$, and the neighborhood (distances) expansion.

In addition, this independent model of mapping higher dimensional mesh to a lower dimensional one can be used in other applications, such as extending the one dimensional non-expansive hashing of Linial and Sasson [20], to arbitrary dimensions (see section 2.8).

2.2 The Model

The following defines several types of paths and cycles on a universe U (either a mesh or torus of arbitrary dimension) with N elements, with the corresponding Manhattan distance d_U (i.e. L^1 norm).

Definition 1 *A* **tour** π *is a one-to-one function* $\pi : U \mapsto \{1, \ldots, N\}$, *where* U *is a mesh (or torus).*

Definition 2 *Let* $\pi : U \mapsto \{1, \ldots, N\}$ *be a one-to-one function, where* U *is a mesh (or torus). Then* π *is called a* **Hamiltonian path** *or shortly a* **path** *if*

$$d_U(\pi^{-1}(x), \pi^{-1}(x+1)) = 1 \qquad \textit{for all } 1 \leq x < N$$

Definition 3 *Let* $\pi : U \mapsto \{1, \ldots, N\}$ *be a Hamiltonian path, where* U *is a mesh (or torus). Then* π *is called a* **Hamiltonian cycle** *or simply a* **cycle** *if*

$$d_U(\pi^{-1}(1), \pi^{-1}(N)) = 1$$

Definition 4 *Let* $\pi : U \mapsto \{1, \ldots, N\}$ *be a one-to-one function, where* U *is a mesh (or torus). Then* π *is called an* $(\boldsymbol{\alpha}, \boldsymbol{\beta})$**-shrinkable numbering** *of vertices if for any two vertices* x *and* y

$$d_U(x, y) \leq \beta(d_1(\pi(x), \pi(y)))^\alpha$$

Remark: The term shrinkable numbering introduced in [11], but similar notions appeared in earlier works, in other contexts. See [19].

We define a variant of paths and cycles, which allows bounded jumps. That is, a tour visiting each node exactly once (Hamiltonicity), and the distance on the mesh (or torus) between each node and its successor on the tour, is bounded by some λ.

Definition 5 *Let $\pi : U \mapsto \{1, \dots, N\}$ be a one-to-one function, where U is a mesh (or torus). Then π is called a* **$\boldsymbol{\lambda}$-bounded jumps path** *or shortly* **$\boldsymbol{\lambda}$-path on $\boldsymbol{U}$** *if*

$$d_U(\pi^{-1}(x), \pi^{-1}(x+1)) \leq \lambda \qquad \textit{for all } 1 \leq x < N$$

Definition 6 *Let $\pi : U \mapsto \{1, \dots, N\}$ be λ-bounded jumps path, where U is a mesh (or torus). Then π is called a* **$\boldsymbol{\lambda}$-bounded jumps cycle** *or shortly* **$\boldsymbol{\lambda}$-cycle on $\boldsymbol{U}$** *if*

$$d_U(\pi^{-1}(1), \pi^{-1}(N)) \leq \lambda$$

Definition 7 *A* **permutation** *π is a one-to-one function $\pi_i : U \mapsto V$, where U, V are meshes (or tori) of arbitrary (possibly different) dimensions, with N elements ($|U| = |V| = N$).*

Definition 8 *Let $\mathcal{F}^{(N)} = \{\pi_1, \pi_2, \dots, \pi_l\}$ be a family of permutations $\pi_i : U \mapsto V$, where U, V are meshes (or tori) of arbitrary (possibly different) dimensions, with N elements ($|U| = |V| = N$). $\mathcal{F}^{(N)}$ is called a* **stereoscopic family of permutations** *w.r.t.* **expansion** *$g_N : \mathbb{N} \mapsto \mathbb{N}$, if*

$$\forall x, y \in U, \qquad \min_i \{\, d_V(\pi_i(x), \pi_i(y)) \,\} \;\leq\; g_N(d_U(x, y))$$

A stereoscopic family of permutations is **good** *if $g_N(d)$ is independent of N and* monotonic *w.r.t. d. For example $g_N(d) = O(d)$ is good and implies that $\mathcal{F}$ preserves distances up to a constant factor.*

Notation: We will denote by d_i the Manhattan distance on the i-dimensional universe (either mesh or torus).

We defined a stereoscopic family of permutations to contain any permutation (one-to-one) functions. Collision-free routing according to a stereoscopic family $\mathcal{F}$ requires particular permutations which are Hamiltonian paths, since at each time step, packets can only be directed to a neighboring node. However, defining the model in a more general way (any one-to-one functions) is more suitable for other applications, such as hashing noisy data (see section 2.8).

The definition of a stereoscopic family of permutations is based solely on the metric defined on the mesh (or torus). No other property of the graph (such as diameter, vertex degree etc.) is considered. In principle, stereoscopic families of permutations can be defined on any graph (network), but this is beyond the scope of the current work.

The model of stereoscopic family of permutations consists of 4 parameters:

δ_1, δ_2 Dimensions of the source and destination meshes.

N Size of domain, namely the number of elements to map.

$|\mathcal{F}|$ Number of permutations (functions) required.

g_N Neighborhood expansion measure, usually considered by its order of magnitude.

We are interested in the asymptotic behavior, where $N \to \infty$. The parameters $|\mathcal{F}|$ and g_N represent the costs (expensive ingredients), and our goal is to reduce these parameters to the minimum. Our study addresses the optimal solutions and the tradeoffs between the two parameters. For example:

- Does constant $|\mathcal{F}|$ suffice ? If so, what is the minimal corresponding g_N ?
- Can g_N be independent of N ?
- Is it possible to achieve $g_N(d) = O(d)$?
- What is the best g_N possible ? What $|\mathcal{F}|$ does it require ?

The study is organized as follows:

Section 2.3 Presents preliminary intuitions to the problem, showing elementary constructions and bounds.

Section 2.4 Presents upper and lower bounds which are tight up to constant factors, for paths in a two dimensional universe.

Section 2.5 Extends the two dimensional upper bound to higher dimensions. That is, a family of $m+1$ paths on the m-dimensional torus.

Section 2.6 Generalizes the upper bound and constructs a stereoscopic family of permutations for mapping any dimensional torus to any other dimensional mesh (or torus).

Section 2.7 Discusses how to use stereoscopic families of permutations for routing, and points out its benefits.

Section 2.8 Provides an example of another application of stereoscopic families of permutations. We generalize non-expansive hashing of one dimensional inputs universe and one dimensional memory (by Linial and Sasson [20]), to inputs universe and memory, each of arbitrary dimension.

2.3 Preliminary Intuitions

2.3.1 Store-and-Forward Algorithm

For comparison, we present an elementary Store-and-Forward algorithm for routing packets in the two dimensional mesh (or torus) in $O({d_p}^2)$ steps. We assume that every node can be the origin of at most one packet.

Packets are routed along an arbitrary shortest path. In case of contention, priority is given to packets whose distance from origin to destination (i.e. d_p) is minimal. So a packet p is delayed only by higher priority packets whose origin is at most d_p steps from p's route. The number of such origins is at most $9{d_p}^2$, as follows. p's route is trivially bounded by a $d_p \times d_p$ square. Extending this square by d_p in each direction, we get a $3d_p \times 3d_p$ square which covers all possible delaying origins.

We assume that every node is the source of at most one packet, so every packet p is delayed at most $9{d_p}^2$ steps (in addition to d_p steps in which it advances). Therefore, every individual packet p reaches its destination within $10{d_p}^2$ steps.

2.3.2 Synchronized Spirals for Batch Routing

A neat static (one shot) hot-potato algorithm to deliver packets in $O({d_p}^2)$ steps on the two dimensional torus is to route each packet in a spiral path (a snail), starting at its origin, as in figure 2.1. In this case, every node can be the source of up to 4 packets, each injected on a spiral of a different direction (orientation).

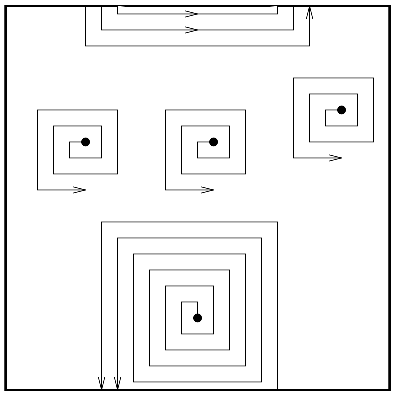

Figure 2.1: Batch routing along spirals

Assume first that every node can be the source of only one packet. If all packets are injected to the network at the same time ($t = 0$), packets routes (spirals) will be synchronized, so that packets never meet (at a vertex), regardless of the network load.

When each node is the source of up to 4 packets, every source node initiates (up to) 4 spirals, one in each direction (orientation): up, down, right and left. Each of the 4 packets is injected to a different orientation spiral. The spiral paths correlate perfectly. At each node, the 4 incoming packets are assigned to distinct outgoing edges, without contention between packets on a link. The exact synchronization between the spirals induces the requested collision-free hot-potato routing.

Every single spiral path covers the whole torus, so each packet will reach its destination within $9{d_p}^2$ steps. Another advantage of the algorithm is that it is not restricted to distinct destinations.

However, the delicate timing is also the algorithm weakness, preventing extension to dynamic routing. In dynamic routing, nodes generate packets continuously, so packets are injected to the network at different times. The spiral paths will then coincide rather than correlate, producing contention on links.

Nodes in the torus belong to different spirals at different times steps. Therefore, the routing table in each node must depend either on the time t, or alternatively, on the source of each incoming packet. Only with this information available, the node can decide how to assign an incoming packet to an outgoing edge. On the other hand, the packet's destination is only important for checking whether the node reached its destination. Further routing decisions at the node (assignments of outgoing edge) are totally independent of the packets' destinations.

2.3.3 One Path in Two Dimensions

Observation 1 *Let M_n be a $n \times n$ mesh (or torus). Then there exists a Hamiltonian path π, such that*

$$\forall x, y \;\; d_1(\pi(x), \pi(y)) \leq 2n \cdot d_2(x, y)$$

where d_1 denotes distance on the (one dimensional) path, and d_2 denotes distance on the (two dimensional) mesh.

Proof. Simply let π scan the mesh (or torus) by rows in a snake like order (see figure 2.2). Let x, y be arbitrary nodes in the mesh M_n. Consider first the case where x, y reside on the same row. Since we scan the whole row at once, their distance on π is at most n.

Otherwise, they are in different rows, in which case the number of rows separating them is at most $d_2(x, y) + 1 \leq 2d_2(x, y)$ (including those of x and y). Therefore, the distance on π between them is at most $n \cdot 2d_2(x, y) \leq 2n \cdot d_2(x, y)$. □

Our goal is a bound for $d_1(\pi(x), \pi(y))$, which is a function of $d_2(x, y)$ and independent of n. The motivation is a very large (possibly infinite) mesh, where nodes communicate with a relatively small neighborhood.

When moving from higher dimensional mesh (or torus) to lower dimensional one, we first encounter the problem of neighborhoods getting smaller. Namely, moving

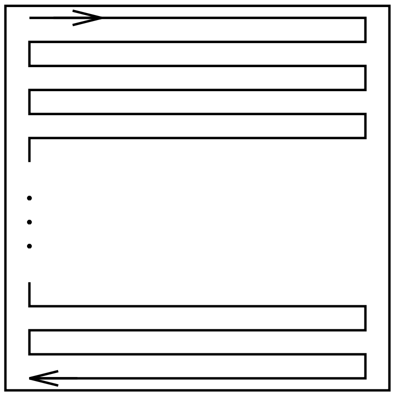

Figure 2.2: Scanning the mesh in a snake like order

from δ_1 dimensions to δ_2 dimensions, the R-neighborhoods shrink from size of $\Theta(R^{\delta_1})$ nodes to the much smaller $\Theta(R^{\delta_2})$ nodes.

Consider a tour on the two dimensional mesh (or torus), which is equivalent to mapping from dimension 2 to 1. The counting argument above shows we must at least square the distances, namely $g(d) = \Omega(d^2)$. Indeed, on the two-dimensional mesh there are $\Theta(R^2)$ nodes in radius R from some fixed node x. So on the line, at least one of these nodes must be $\Omega(R^2)$ steps far from x.

In fact, the difficulty of mapping from two dimensions to one dimension is much more profound than the sizes of the neighborhoods. The following lemma shows that even quadratic expansion of distances does not suffice.

Lemma 2 *Let π be a Hamiltonian path (or cycle) on the $n \times n$ two dimensional mesh (or torus). Then*

$$\exists x, y \quad d_2(x,y) = 1 \ \wedge \ d_1(\pi(x), \pi(y)) > \frac{n}{4}$$

where d_1 denotes distance on the (one dimensional) path, and d_2 denotes distance on the (two dimensional) mesh.

Proof. Since the path (or cycle) π is of size n^2, there is a pair of nodes whose distance on the path is at least $\frac{n^2}{2}$. Let x', y' be the corresponding nodes in the mesh, so $d_1(\pi(x'), \pi(y')) \geq \frac{n^2}{2}$. Recall, however, that the distance on the torus (or mesh) is bounded by $2n$, therefore there is a path from x to y on the two dimensional torus $x' = x_1, x_2, \ldots, x_r = y'$ for some $r \leq 2n$.

Assume to the contrary, that $d_1(\pi(x_i), \pi(x_{i+1})) \leq \frac{n}{4}$ for all i. Then we can bound the overall distance between $\pi(x_1)$ and $\pi(x_r)$ using the triangle inequality:

$$d_1(\pi(x'), \pi(y')) = d_1(\pi(x_1), \pi(x_r)) \leq \sum_{i=1}^{r-1} d_1(\pi(x_i), \pi(x_{i+1})) \leq (2n-1) \cdot \frac{n}{4} < \frac{n^2}{2}$$

Contradiction. □

This lemma implies that g_N, the expansion of distances, cannot be defined independently of N, even not as an exponential function or so. Moreover, there will always be nodes whose distance expands from 1 on the mesh, to $\frac{n}{4}$ on the line (a huge expansion in terms of $d = 1$).

Nevertheless, the straightforward construction yields quite a tight upper bound for one path on the two dimensional mesh, indicating we reached the limits of the model. We conclude that representing several dimensions requires a more powerful model, such as the stereoscopic family of permutations.

2.3.4 Two Paths in Two Dimensions

The model of stereoscopic family of permutations defined in section 2.2 allows us to use several Hamiltonian paths in order to keep the neighborhoods expansion small. Indeed, we can extend the single path constructed in Observation 1, to a family of two paths with better performance (smaller distances expansion g_N). The performance of these two specific paths is better than any single path, according to the lower bound of Lemma 2.

Lemma 3 *Let M_n be a $n \times n$ mesh, where n is odd. Then there exist 2 Hamiltonian paths π_1, π_2 on M_n, such that*

$$\forall x, y \ \min_{i=1,2}\{d_1(\pi_i(x), \pi_i(y))\} \leq \begin{cases} 4\sqrt{n} \cdot d_2(x,y) & \text{if } d_2(x,y) < \frac{1}{8}\sqrt{n} \\ 4n \cdot d_2(x,y) & \text{otherwise} \end{cases}$$

Implying

$$\forall x, y \ \min_{i=1,2}\{d_1(\pi_i(x), \pi_i(y))\} \leq 32\sqrt{n} \cdot (d_2(x,y))^2$$

where d_1 denotes distance on the path (one dimensional), and d_2 denotes distance on the mesh (two dimensional).

Proof. Intuitively, we partition M_n into $\sqrt{n}$ horizontal strips, each of height $\sqrt{n}$. The path π_1 scans T_n strip by strip, where in each strip traversal is by columns. π_2 is like π_1, but shifted half a strip downwards.

Formally, we pick a number $s \in [\sqrt{n}, 2\sqrt{n}]$, and partition M_n into s (almost) equal strips (each of size $\sim \frac{n}{s}$). To be precise, suppose $n = sq + r$ where $0 \leq r < s$ and $q = \lfloor \frac{n}{s} \rfloor$, then the first r strips consist of $q + 1$ rows, and the remaining $s - r$ strips consist of q rows. The "height" of a single strip is bounded by:

$$q + 1 \leq \frac{n}{s} + 1 \leq \sqrt{n} + 1 \leq 2\sqrt{n}$$

Path π_1 travels M_n strip by strip as follows. It enters the first strip at the Upper-Left corner, and traverses the strip by columns towards the right hand-side, i.e. it goes down the first column, climbs up the second column, then down the third column, and so on (see figure 2.3). Since the number of columns is odd, the strip traversal ends up in the Lower-Right corner of the strip, and π_1 enters the second strip at its Upper-Right corner. It travels this strip towards the left hand-side (by columns, till the Lower-Left corner) and so on, strip by strip, from top to bottom.

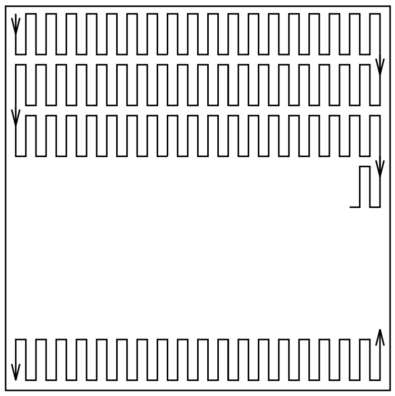

Figure 2.3: Scanning the mesh by strips

π_2 is defined to be a shift of π_1 for $\lfloor \frac{q}{2} \rfloor$ units (half the strip size) down. For this purpose, the "height" of the first and last strips is half the height of other strips.

Let x, y be arbitrary nodes in the mesh, and we will bound their distance on the π_i's. Consider first the case that for some path π_i they are in the same strip. Then the horizontal distance (distance between their columns) is bounded by $d_2(x, y)$. Since π_i scans this strip by columns, we get

$$d_1(\pi_i(x), \pi_i(y)) \leq (d_2(x, y) + 1) \cdot (q + 1) \leq 4\sqrt{n} \cdot d_2(x, y)$$

Otherwise, the two nodes are in different strips in both paths, and thus the distance between them must be at least

$$d_2(x, y) \geq \frac{1}{2}q \geq \frac{1}{2}(\frac{n}{2\sqrt{n}} - 1) \geq \frac{1}{8}\sqrt{n}$$

However, the number of strips between them is at most $\frac{d_2(x,y)}{q} + 1$ (including those of x and y), so

$$d_1(\pi_i(x), \pi_i(y)) \leq (q + 1)n \cdot (\frac{d_2(x, y)}{q} + 1) \leq 4n \cdot d_2(x, y) \leq 32\sqrt{n} \cdot (d_2(x, y))^2$$

□

Lemma 4 *Let T_n be a $n \times n$ torus, where n is even. Then there exist 2 Hamiltonian cycles π_1, π_2 on T_n, such that*

$$\forall x, y \ \min_{i=1,2}\{d_1(\pi_i(x), \pi_i(y))\} \le 64\sqrt{n} \cdot (d_2(x,y))^2$$

where d_1 denotes distance on the (one dimensional) cycle, and d_2 denotes Manhattan distance on the (two dimensional) torus.

Proof. Intuitively, as in the mesh, we partition T_n into $\sqrt{n}$ horizontal strips, each of width $\sqrt{n}$. π_1 scans T_n strip by strip, where in each strip, the traversal is by columns. However, due to the wrap-around nature of the torus, the strips will be slanted, so that the strips are connected to each other in the wrap-around (the strip slope is $\frac{\sqrt{n}}{n}$). π_2 is half a strip downwards shift of π_1.

Formally, we pick a number $s \in [\sqrt{n}, 2\sqrt{n}]$, and partition T_n into s (almost) equal strips (each of size $\sim \frac{n}{s}$). Suppose $n = (2s)q + r$ where $0 \le r < 2s$ and $q = \lfloor \frac{n}{2s} \rfloor$. Since n is even, r is also even, and $n = s(2q) + r$. The first $\frac{r}{2}$ strips consist of $2q + 2$ rows, and the remaining $s - \frac{r}{2}$ strips consist of $2q$ rows. The "height" of a single strip is bounded by:

$$2q + 2 \le 2\frac{n}{2s} + 2 \le \sqrt{n} + 2 \le 3\sqrt{n}$$

The strips are slanted downwards using "steps" (see figure 2.4). The height of each step is 2, and the steps are in specific $q+1$ columns which are spread uniformly on the strip (every $\sim \frac{n}{q+1}$ nodes). Clearly, in $s - \frac{r}{2}$ strips q steps suffice, so only the first q of the $q+1$ columns will be used.

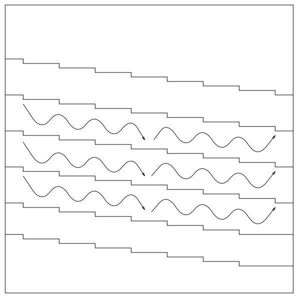

Figure 2.4: Slanted strips on the Torus

π_1 is defined to travel T_n strip by strip, each of them traversed by columns. Upon reaching a step, the path might deviate from columns scanning, as in figure 2.5. The strips are concatenated (in the wrap-around), forming a one long strip, in which π_1 travels. π_1 is actually a cycle, because the number of columns is even.

Define π_2 to be a q units (half the strip height) downwards shift of π_1.

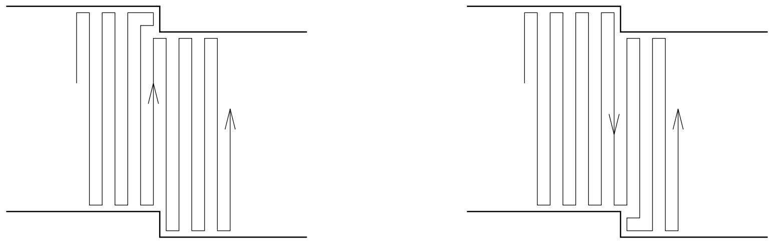
Figure 2.5: Deviation from columns scanning on strip steps

Let x, y be arbitrary nodes in the mesh, and we will bound their distance on the π_i's. Consider first the case that for some path π_i they are in the same strip. Then the horizontal distance (distance between their columns) is bounded by $d_2(x, y)$. Since π_i scans this strip by columns (up to 2 columns deviation), we get

$$d_1(\pi_i(x), \pi_i(y)) \leq (d_2(x, y) + 2) \cdot (2q + 2) \leq 9\sqrt{n} \cdot d_2(x, y)$$

Otherwise, the two nodes are in different strips in both paths, and thus the distance between them must be at least

$$d_2(x, y) > \frac{1}{2}q \geq \frac{1}{4}\frac{n}{2s} \geq \frac{1}{16}\sqrt{n}$$

However, the number of strips between them is at most $\frac{d_2(x,y)}{2q} + 1$ (including those of x and y), so

$$d_1(\pi_i(x), \pi_i(y)) \leq (2q + 2)n \cdot (\frac{d_2(x, y)}{2q} + 1) \leq 4n \cdot d_2(x, y) \leq 64\sqrt{n} \cdot (d_2(x, y))^2$$

$\square$

2.3.5 Preserving Only Specific Distances

We stress that the difficulty in preserving the metric is the global handling of all possible pairs at different possible distances. Specific distances (neighborhoods) such as $\Delta = 5$ or $\Delta = \sqrt{n}$ are maintained by two paths. The construction is similar to section 2.3.4, with strips adjusted according to the distance we want to preserve.

Suppose that we want to preserve a distance $\Delta(n)$ on the mesh. We follow the proof of Lemma 3, with strips of "height" 2Δ (roughly), and π_2 is a shift of π_1 by half a strip "height" (see figure 2.3). Every two nodes at distance Δ (or less), must be in the same strip in one of the paths π_i. The distance of the two nodes on this path π_i, is then bounded by $O(\Delta^2)$.

2.4 Two dimensions

Recall that our goal is to bound the neighborhood expansion g_N independently of N. However, the lower bound following from Lemma 2 shows this is impossible unless using at least two paths. Although our constructions with two paths are better than the lower bound for one path, they still expand neighborhoods significantly.

In this section we consider the two dimensional case, and present a family $\mathcal{F}$ of 3 Hamiltonian paths, with distances expansion $g(d) = O(d^2)$. In addition, we prove this is optimal with respect to both the number of paths and the order of magnitude of the expansion g.

2.4.1 Quadratic Expansion Lower Bound

Observation 5 *Let $\{\mathcal{F}^{(N)}\}_{N=1}^{\infty}$ be stereoscopic families of permutations on the two dimensional mesh (or torus), and let $|\mathcal{F}^{(N)}| = O(1)$. Then $g_N(d) = \Omega(d^2)$.*

Proof. Consider the following counting argument. Let x be a fixed node on the two dimensional mesh (or torus). Then the number of nodes on the mesh (or torus) in radius R from x is quadratic, namely $\Theta(R^2)$. Since the number of paths in $\mathcal{F}$ is constant, at least one of these nodes must be $\Omega(R^2)$ steps far from x on all paths. $\square$

2.4.2 Constructive Upper Bound with Three Paths

Following we construct a stereoscopic family of permutations $\mathcal{F} = \{\pi_1, \pi_2, \pi_3\}$ with expansion $g(d) = O(d^2)$, which is the best possible expansion up to constant factors, by Observation 5. We will see later (section 2.4.5) that it is optimal w.r.t. $|\mathcal{F}|$. The most natural graph for this construction is a $n \times n$ torus, where $n = 3 \cdot 2^l$, yielding simpler proof and lower constants (Lemma 6 below). Adjustments to arbitrary tori and meshes are possible (with larger constants), and are discussed in sections 2.4.3 and 2.4.4.

Lemma 6 *Let T_n be a $n \times n$ torus, where $n = 3 \cdot 2^l$, for arbitrary $l > 0$. Then there exists a family $\mathcal{F} = \{\pi_1, \pi_2, \pi_3\}$ of 3 Hamiltonian cycles on T_n, such that*

$$\forall x, y \min_{i=1,2,3} \{d_1(\pi_i(x), \pi_i(y))\} \leq 36(d_2(x, y))^2$$

where d_1 denotes distance on the (one dimensional) cycle, and d_2 denotes Manhattan distance on the (two dimensional) torus.

Proof. First denote each torus node as a pair (i, j) where $1 \leq i, j \leq n$. Following, we construct a Hamiltonian cycle π, similar to the one used by Lempel and Ziv for compressing two-dimensional data, in [19]. Consider the torus T_n as a $n \times n$ mesh M, and we shall show a Hamiltonian path on M from $(1,1)$ to $(n,1)$. This Hamiltonian path on M is defined recursively, by partitioning M to 2 segments in each axis, to get 4 equal size sub-meshes M_1, M_2, M_3, M_4, as in figure 2.6. The path starts at $(1,1)$ which is in M_1, and ends up at $(n,1)$ which is in M_4. Therefore, the path shall be $M_1 \to M_2 \to M_3 \to M_4$, as in figure 2.6.

Constructing the path inside M_1 is done by breaking M_1 into 4 sub-meshes M_{11}, M_{12}, M_{13} and M_{14}. The path has to travel from M_{11}, recursively cover all M_1 sub-meshes, and finally move to M_2. The only way to do it (see figure 2.6), is $M_{11} \to M_{14} \to M_{13} \to M_{12}$.

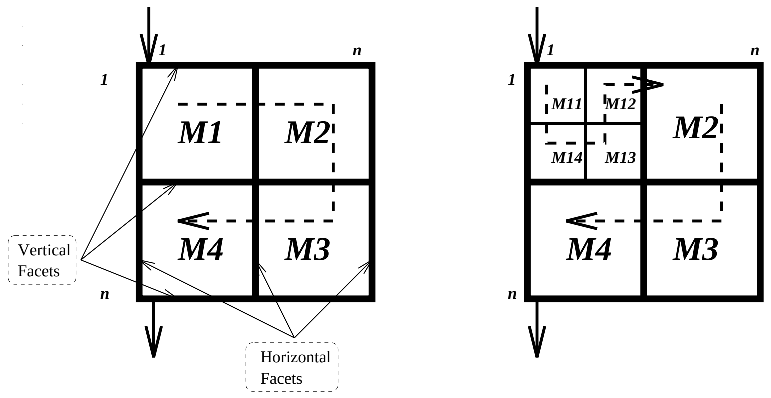

Figure 2.6: The Hamiltonian path and Refining it inside M_1

Formally, at each phase we break our mesh M into 2 segments at each axis, get 4 equal size sub-meshes M_1, M_2, M_3, M_4, and consider this high-level description as a 2×2 mesh. We define a ***d*-facet** to be a virtual border line between adjacent $d \times d$ sub-meshes (i.e. face/edge of the square, see figure 2.6).

By definition, transitions between successive sub-meshes are made only through the sub-meshes corner nodes (i.e. one of $(1,1); (1,n); (n,1); (n,n)$). In addition, in each sub-mesh, the entrance (corner) node and the exit (corner) node are necessarily adjacent corners. W.l.o.g. we assume the entrance node is $(1,1)$, and thus the exit node is either $(1,n)$ or $(n,1)$. A corresponding Hamiltonian path is always feasible by recursion, as shown in figure 2.7.

The path is constructed recursively, until we get down to a 3×3 mesh, in which a similar construction is also feasible (see figure 2.8).

Basic path properties:

1. After i recursion phases, mesh M_n is partitioned into 2^i equal segments in each axis, total of $(2^i)^2 = 2^{2i}$ sub-meshes, each of size $\frac{n}{2^i} \times \frac{n}{2^i}$.

2. Each of the construction sub-meshes is traversed as a whole (i.e. it is covered sequentially without any jumps outside).

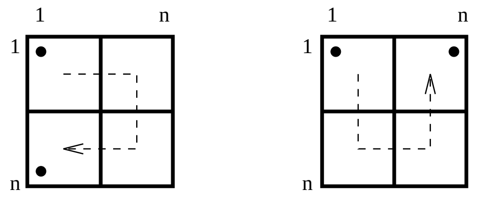

Figure 2.7: Demonstrating feasibility of 2×2 path

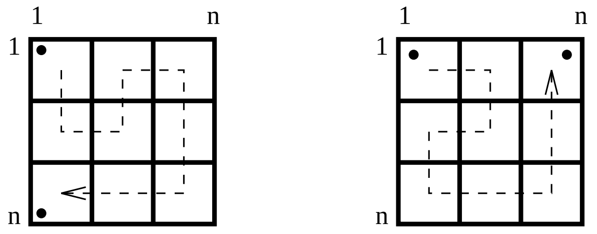

Figure 2.8: Feasibility of 3×3 Hamiltonian paths

3. The distance on π of any two nodes in the same sub-mesh, say M' from the i-th phase, is less then the size of the sub-mesh, i.e. $d_1(\pi(x), \pi(y)) < (\frac{n}{2^i})^2$

4. The path visits each node once, and not more. Therefore, it is Hamiltonian.

5. The path starts in $(1,1)$ and ends up in $(n,1)$, which are neighbors in the original torus T_n. Hence, π is a Hamiltonian cycle on the torus.

We define π_1 to be exactly π, as described above. Since the torus is cyclic, any path on it can be moved (shifted) using an additive transformation modulo n. So let π_2 be the same cycle as π shifted $\frac{n}{3}$ units in each axis (say right and down), and π_3 the same with $\frac{2n}{3}$ units shift. This can be alternatively viewed as fixing the mesh (on the torus) with a $\frac{n}{3}$ (or $\frac{2n}{3}$) shift in each axis, and then constructing the path recursively.

Let x, y be arbitrary nodes on the torus, and we will bound their distance on the π_i's. Denote $d' = d_2(x,y)$ their distance on the torus, and let $d = 3 \cdot 2^j$ be the smallest such that $d > 3d'$, so clearly, $d \leq 6d'$.

We now partition the torus into sub-meshes of size $d \times d$ (i.e. by $\log_2 \frac{n}{d} = l - j$ phases of the construction), once for each path π_i (totally 3 sets of sub-meshes).

Let f_1 be a horizontal facet in the $d \times d$ partitioning of π_1, and f_2 a horizontal facet in the $d \times d$ partitioning of π_2. We claim that the distance between these facets is at least $\frac{d}{3}$ mesh nodes. Indeed, the partitioning of n to $\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \ldots$ does not coincide with the shift by a $\frac{1}{3}$, and n is chosen to be divisible by all these denominators. Specifically, f_1 appears every d rows, and f_2 appears at $\frac{n}{3}$ plus multiples of d. Consider the location of f_2 relatively to f_1, i.e. its location modulo d (since f_1 appears at multiples of d). It comes down to resolve $\frac{n}{3}$ modulo d, and $\frac{n}{3} = 2^l = d \cdot \frac{2^{l-j}}{3}$. Since 3 does not divide 2^{l-j}, we conclude that the difference between adjacent f_1 and f_2 is $\frac{d}{3}$.

By symmetry considerations, we conclude an important property that any two d-facets of different cycles ($\pi_i \neq \pi_{i'}$) but with the same orientation (either horizontal or vertical) are separated by at least $\frac{d}{3}$ torus nodes.

We now claim that x, y belong to the same $d \times d$ sub-mesh in some Hamiltonian cycle π_i, and hence their distance on this π_i is less than the sub-mesh size, and we're done:

$$d_1(\pi_i(x), \pi_i(y)) < d^2 \leq (6d')^2 = 36(d_2(x,y))^2$$

Assume to the contrary, that x and y are in different $d \times d$ sub-meshes in all 3 Hamiltonian cycles. Thus, x, y are separated from each other by at least one facet in any π_i. These 3 facets are either horizontal or vertical, so at least two must be of the same orientation (by the pigeon-hole principle), w.l.o.g. say horizontal. Therefore, moving from x to y, one must cross these two horizontal facets, which are separated from each other by at least $\frac{d}{3}$ rows, so $d_2(x,y) \geq \frac{d}{3} > d' = d_2(x,y)$. Contradiction.

□

The specific paths used in this construction have yet another property which is complementary to our requirement of bounding the neighborhoods expansion. In each path, distances on the path shrink to their square roots when placed on the torus.

Lemma 7 *Let π be the path constructed recursively in the proof of Lemma 6. Then π is a $(\frac{1}{2}, 6)$-shrinkable numbering of vertices. That is*

$$\forall x, y \qquad d_2(x,y) \leq 6\sqrt{d_1(\pi(x), \pi(y))}$$

Proof. Denote $d = d_1(\pi(x), \pi(y))$. Then there is some $d' \times d'$ square in the construction such that $\sqrt{d} \leq d' < 2\sqrt{d}$. Assume w.l.o.g. that x appears before y on the path. Then traveling on the path from x to y (total of $d-1$ steps), one can either stay in the same $d' \times d'$ square or advance to a neighboring one, but not further (because the square's size is $(d')^2 \geq d$). Either case, x and y are in the same $2d' \times d'$ rectangle on the mesh, and thus

$$d_2(x,y) \leq d' + 2d' \leq 6\sqrt{d}$$

□

In other words, $\mathcal{F}$ always expands distances at least to their quadratics. Therefore, routing along the paths would always require $\Theta({d_p}^2)$ steps.

2.4.3 Extending the Upper Bound to Arbitrary Meshes

Lemma 6 constructs three paths on the $n \times n$ torus, where $n = 3 \cdot 2^l$. This construction can be adjusted to support any rectangular ($n_1 \times n_2$) mesh. It will be less elegant and the constants become larger. We push the problem to the mesh sideways, by building the original construction in the center (with $d \times d$ squares for all $d = 3 \cdot 2^j$), and adjusting each path in its marginal rectangles, as follows.

The important property to maintain is that facets of each path π_i will appear exactly every $3 \cdot 2^j$ nodes (for all j). This will guarantee distance of $\frac{d}{3}$ between d-facets of different paths. Thus, any pair of nodes x, y, whose is distance $d' = d_2(x, y)$, will still be connected by some π_i within $(6d')^2$.

For each path π_i we first draw the facets lines for every possible $d = 2^j$, starting from the upper left corner (plus the appropriate shift). This creats $d \times d$ squares for d's at all levels. Figure 2.9 shows an example for setting the facets on a rectangle. Secondly, we build the path π_i inside each square, recursively. In the marginal rectangles, it might happen that the d-facets create a rectangle, not completing a $d \times d$ square because of the mesh ends. Figure 2.9 illustrates the path construction, and several rectangles are further refined.

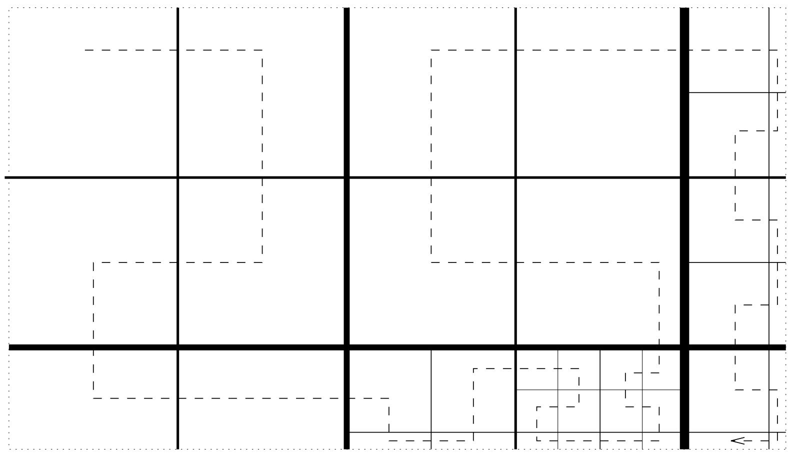

Figure 2.9: Creating facets exactly at $d = 3 \cdot 2^j$ nodes

The path π_i can still be constructed recursively according to our original guidelines. Consider a rectangle bounded by d-facets on some sides and the mesh ends on others, and check how it is divided by the $\frac{d}{2}$-facets:

- If no $\frac{d}{2}$-facet appears inside the rectangle, we do nothing and build the path recursively (the d-facets are also $\frac{d}{2}$-facets).
- If only one $\frac{d}{2}$-facet appears inside the rectangle, only one exit node is reasonable, and the path can be constructed recursively, as in figure 2.10.

- If two $\frac{d}{2}$-facets appear inside the rectangle, they partition it to 4 sub-rectangles and we can follow the original scheme of recursing into each one of them, through adjacent entrance and exit corners. Two example are shown in figure 2.11.

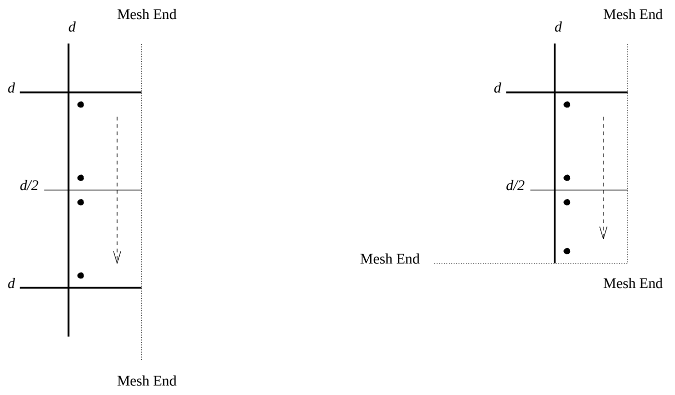

Figure 2.10: Rectangles with 1 internal $\frac{d}{2}$-facet

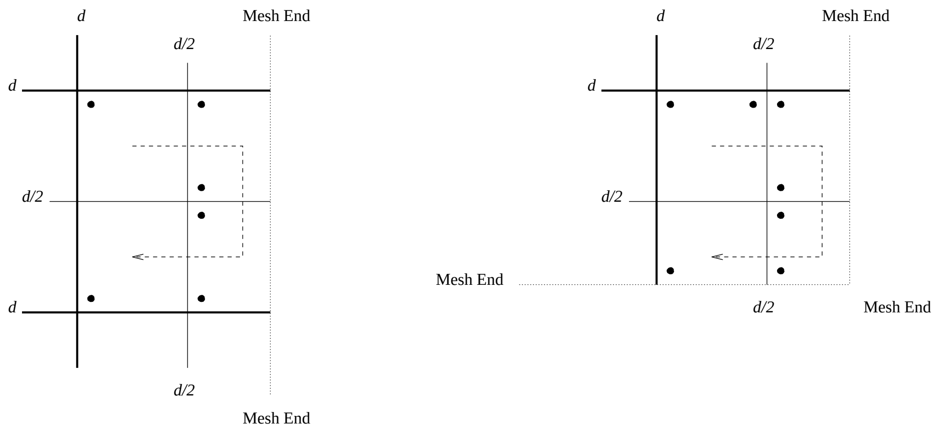

Figure 2.11: Rectangles with 2 internal $\frac{d}{2}$-facets

At the lowest level, the required Hamiltonian paths (with entrance and exit nodes at adjacent corners) can be constructed, unless the rectangle is of the type $odd \times even$ (or $even \times odd$). Since we have facets every 3 nodes, this can only happen at 3×2 rectangles and 1×2 rectangles. For example, if 3 divides n this rectangles do no occur. Anyway, on this lowest level, local fixes can be made, and the constants might increase.

2.4.4 Extending the Upper Bound to Arbitrary Tori

Torus graphs require different adjustments. The concept used to adjust the paths in meshes fail in the wrap-around, because d-facets from different paths might be close to each other (closer than $\frac{d}{3}$ nodes). Lemma 6 has to be adjusted in a different approach to support arbitrary $n \times n$ tori.

The important property for torus graphs is to keep the facets equally spread all over the n nodes, so going through the wrap-around will be smooth. We define the facets at the j-th phase of the recursion to appear after approximately (it might be non-integer) $\frac{n}{2^j}$. That is, the i-th facet will appear after $\lfloor i\frac{n}{2^j} \rfloor$. First note that the distance between successive facets of the same phase (in the same path) is more or less uniform ($\frac{n}{2^j} \pm 1$). Second, each phase refines the previous one, so facets of the j-th phase are also facets in phase $j+1$, and each interval is partitioned to 2. Third and most important, the facets are aligned to the center, and not shifted to the sideways. For example, the facet of the first phase is right in the middle, and not shifted to the sides, as might happen when concentrating all the small facets in one side. So 14 is divided to 7+7, which in turn is divided to 3+4+3+4. Had we divided it to 3+3+4+4, the $\frac{1}{2}n$ (middle) facet would be shifted from the center to the left (see figure 2.12).

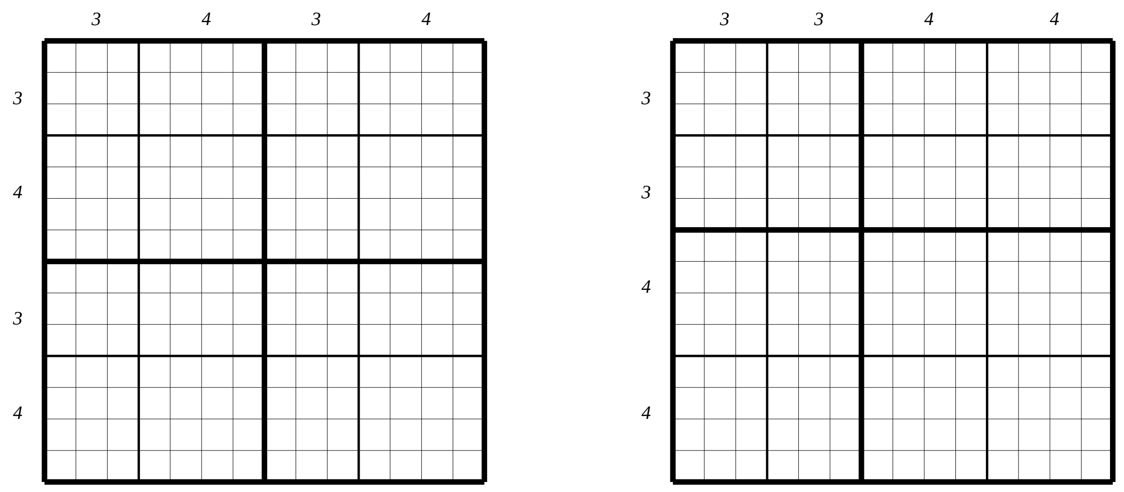

Figure 2.12: Example of aligned facets (on the left) and not-aligned (on the right).

So the torus is divided to 4 rectangles whose sides are more or less equal ($\sim \frac{n}{2^j}$). Each rectangle is divided in turn to 4 smaller rectangles, whose sides are also more or less equal, and so on. The path is constructed recursively in each of the 4 sub-rectangles, as in the original construction of π in Lemma 6 (see figure 2.6). The recursion stops at rectangles whose both sides are smaller than 4. Rectangles of the type *odd* $\times$ *even* (and *even* $\times$ *odd*) do not necessarily support a Hamiltonian path between adjacent corners (our entrance and exit nodes). Local fixes are made by taking another corner to be the entrace (or exit) node. The neighboring rectangle has to be altered accordingly, see figure 2.13 for illustration. The local fixes might be required on all levels, so we do them in a top down fashion, starting with the largest "bad" rectangles (*odd* $\times$ *even* or *even* $\times$ *odd*).

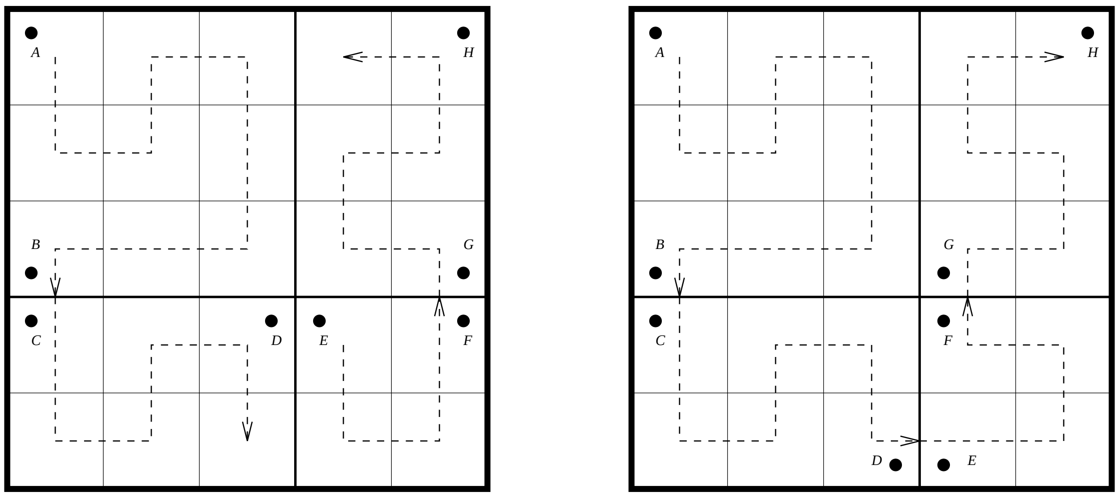

Figure 2.13: Example of bad rectangles (see nodes D and H on the left) and adjusting them (on the right)

The paths π_1, π_2, π_3 are constructed by shifting the path π by $\lfloor \frac{n}{3} \rfloor$ in each axis, as in the original proof. Since the facets occur at $\sim \frac{n}{2}, \sim \frac{n}{4}, \sim \frac{n}{8}, \sim \frac{n}{16}, \ldots$ and the paths are shifted by $\sim \frac{n}{3}$, two d-facets of different paths but with the same orientation (horizontal or vertical), are at least $\sim \frac{d}{3}$ far from each other. Therefore, the proof of Lemma 6 still holds, with larger constant factors.

Similar methods can be applied to extend the construction to rectangular tori of the type $n \times \alpha n$. The torus is divided not to 4 smaller rectangles, but rather to $2 \times \hat{\alpha}$ rectangles. One would like $\hat{\alpha} = \sim \alpha$ to be even in order to avoid problems with entrance and exit nodes at adjacent corners, and so on.

2.4.5 Three Paths Lower Bound

We saw that three paths suffice to preserve quadratic distances (see section 2.4.2), but a single path is definitely not enough (see section 2.3.3). We will now address the question of preserving quadratic distances with two paths.

Definition 9 *Let π be a path. A* **segment s of π** *is a restriction of π resulting with the image set $[a, b]$, i.e. π restricted to the domain $\pi^{-1}([a, b])$.*

Observation 8 *Let s be a segment of path π on the $n \times n$ mesh M_n (or torus T_n where $|s| < n$). Then*

1. *The projection of the path on every single axis is a segment.*
2. *At least one of these segments is of size $\geq \sqrt{|s|}$.*

Proof.

1. Since the path is "continuous" (cannot jump more than 1 unit at a time), the projection on any axis is a segment.
2. Assume these segments (projections on x and y) are of sizes a and b, respectively, then clearly $ab \geq |s|$. Hence $\max\{a, b\} \geq \sqrt{|s|}$.

$\square$

Definition 10 *Let U be a $n \times n$ mesh or torus universe. Then $\pi : U \mapsto \{1, 2, \ldots, n^2\}$ is a* **diagonal neighbors path** *if for all i, $\pi^{-1}(i)$ and $\pi^{-1}(i+1)$ are either neighbors or diagonal neighbors (see figure 2.14 for illustration).*

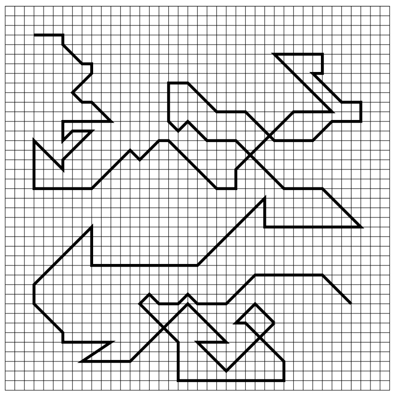

Figure 2.14: Diagonal neighbors path

Lemma 9 *Let T_n be the $n \times n$ torus (or mesh), and π_1, π_2 be Hamiltonian paths on T_n. Then for any constant $c > 0$ and sufficiently large n,*

$$\exists x, y \qquad \min_{i=1,2}\{d_1(\pi_i(x), \pi_i(y))\} > c(d_2(x, y))^2$$

where d_1 denotes distance on the (one dimensional) cycle (or line), and d_2 denotes Manhattan distance on the (two dimensional) torus (or mesh).

Sketch of Proof. We first consider a segment of π_1, and show that it must have some "boundary" (frontier), which is wide enough (see figure 2.15). This path, π_1, does not connect nodes in the frontier to their neighbors just outside the frontier with a quadratic expansion. Thus, the other path, π_2, must handle these pairs. It follows that π_2 must oscillate along this boundary, and cannot get too far from it in this section (the oscillations have a small amplitude). We then take a node x in the center of the frontier and a node y just beyond the oscillations of π_2. It follows that both paths visit x but do not visit y in this section, so neither path handles the pair x, y.

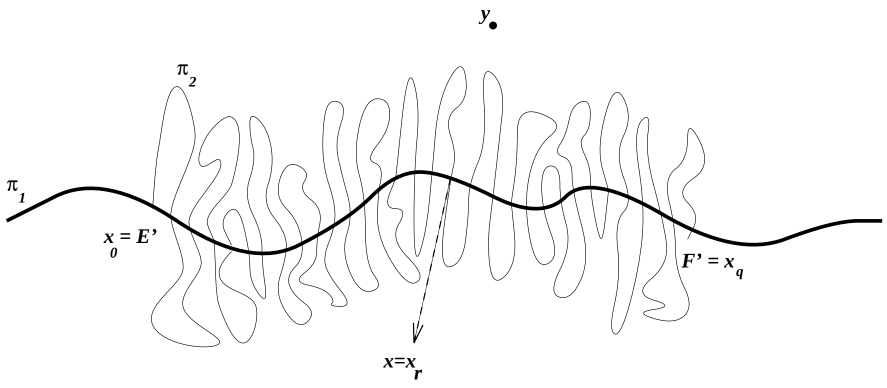

Figure 2.15: π_2 oscillates along the border of π_1.

Proof. Assume n is large enough, so that for example $\frac{1}{4}\sqrt[4]{n} > c$ and w.l.o.g. $c \geq 1$. From now on, we deal only with respectively small objects (segments of paths, etc.), up to size $\frac{1}{2}n$. Therefore we can consider our torus to be a mesh, assuming w.l.o.g. that the following construction takes place in the center of the mesh.

Let s_1 be a segment of length $k = \sqrt{n}$ of π_1 (in the center of the mesh), whose endpoints are E, F. W.l.o.g. the projection of π_1 on the x-axis (horizontal) is larger than on the y-axis. Denote this projected interval by I, thus $|I| \geq \sqrt{k}$. Let I_E, I_F be $\frac{1}{10}\sqrt{k}$ intervals around the x-axis projection of E and F, respectively. Removing I_E and I_F from I, we get at most 3 sub-intervals of total size $|I \setminus (I_E \cup I_F)| > \frac{1}{2}|I|$, so at least one sub-interval I' is of size $|I'| > \frac{1}{3}(\frac{1}{2}|I|) \geq \frac{1}{6}\sqrt{k}$. Denote this interval as $I' = [a, b]$, and let E', F' be the highest nodes in the columns a and b, respectively, which are on s_1 (see figure 2.16). E', F' define a sub-segment of s_1, denote it as t_1. Let y be a node in the central column of I', above t_1 in this column and whose distance from t_1 is $l = 6c$. Trivially, such y must exist and let x be its closest node on t_1, so $d_2(x, y) = l$.

We now want to find the boundary (contour) of t_1 w.r.t. y (i.e. from above). In order to do so we define a border line R starting at E' and going $2k$ nodes vertically

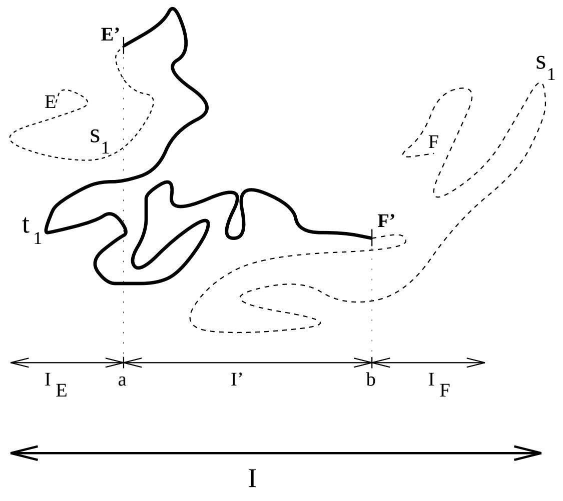

Figure 2.16: Removing endpoints intervals I_E, I_F from s_1, we get t_1 and $I' = [a, b]$

upwards, then horizontally right (or left) until the column of F', and then vertically down to F' (see figure 2.17). Let $C = C_y(t_1)$ be all nodes reachable from y without using (crossing) nodes from the border line R nor the path t_1.

Let $B = B_y(t_1)$ be all nodes in t_1 which have a neighbor in C. We claim that B is a diagonal neighbors path connecting E' and F'. Indeed, consider the nodes as squares in a continuous plane, then C is actually a connected set of these squares (add the squares one by one), whose boundary is thus a closed polygonal line. In particular, this line connects E' and F' and corresponds to B above, proving B's nature (see figure 2.18).

We also claim that $x \in B$ as follows. By definition, $x \in t_1$. Consider a shortest path from x to y. No node in this path is in t_1, or otherwise the distance from y to t_1 would be less than $l = d_2(x, y)$. No node in this path is neither in R because the boundary R is much further than l from y. For example, $l = 6c << \Omega(\sqrt{k}) \leq \frac{1}{2}|I'|$. Therefore, every single node in this path is in C, and thus x has a neighbor in C.

Consider now going from E' to F' along B and visiting x on the way, denoted as $E' = x_0, x_1, x_2, \ldots, x = x_r, \ldots, x_q = F'$. Recall B is a diagonal neighbors path, thus $d_2(x_j, x_{j+1}) \leq 2$. By definition of B, any x_j has a neighbor $x'_j \in C$, and thus x'_j is not in s_1.

Note that $d_2(x'_j, x_{j+1}) \leq 3$, so they must be connected by some π_i within $c \cdot 3^2 = 9c$ steps. But x'_j is not on s_1, so their distance on π_1 must be at least $\frac{1}{10}\sqrt{k} > 9c$. Therefore, their distance on π_2 must be at most $9c$. The same argument applies also to x_j and x'_j, which are neighbors and therefore must be connected by π_2 with distance $c \cdot 1^2 = c$. Hence,

$$d_1(\pi_2(x_j), \pi_2(x_{j+1})) \leq c + 9c = 10c$$

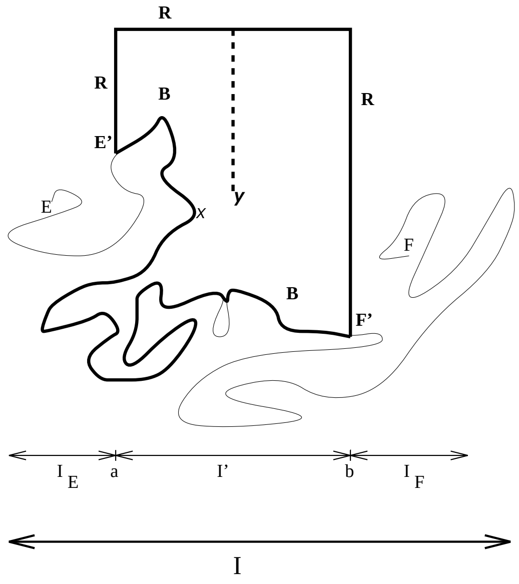

Figure 2.17: Border line R and boundary $B_y(t_1)$

Consider now π_2 visiting $E' = x_0$. It must get to x_1 within $10c$ steps, to x_2 within the following $10c$ steps and so on, until reaching $x_q = F'$ (see figure 2.15). Therefore, the maximal distance it can get far from t_1 is $5c < l$. So although π_2 visits $x = x_r$, it cannot visit y in this section (from x_0 to x_q), and thus the distance between x and y on π_2 is at least $\frac{1}{4}|I'|$ (x is close to the central column of I').

The distance of x and y on π_1 is also at least $\frac{1}{4}|I'|$. However, at least one of the paths must connect x, y within $c \cdot l^2 = 36c^3 << \Omega(\sqrt{k}) \leq \frac{1}{4}|I'|$ steps. Contradiction. $\square$

The proof method used in this lower bound, can be enhanced to any expansion function, and not only a quadratic one. Improving the lower bound to hold for arbitrary expansion $g : \mathbb{N} \mapsto \mathbb{N}$, implies that n is necessary and no bound of d alone holds for family of two Hamiltonian paths.

Corollary 10 *Let T_n be the $n \times n$ torus (or mesh), and π_1, π_2 be Hamiltonian paths on T_n. Then for any expansion function $g : \mathbb{N} \mapsto \mathbb{N}$ and sufficiently large n,*

$$\exists x, y \qquad \min_{i=1,2}\{d_1(\pi_i(x), \pi_i(y))\} > g(d_2(x, y))$$

where d_1 denotes distance on the (one dimensional) cycle (or line), and d_2 denotes Manhattan distance on the (two dimensional) torus (or mesh).

Proof. Follow the proof of Lemma 9. Once again we assume n is large enough, so $\sqrt[4]{n} << \sqrt{n}$ and so on.

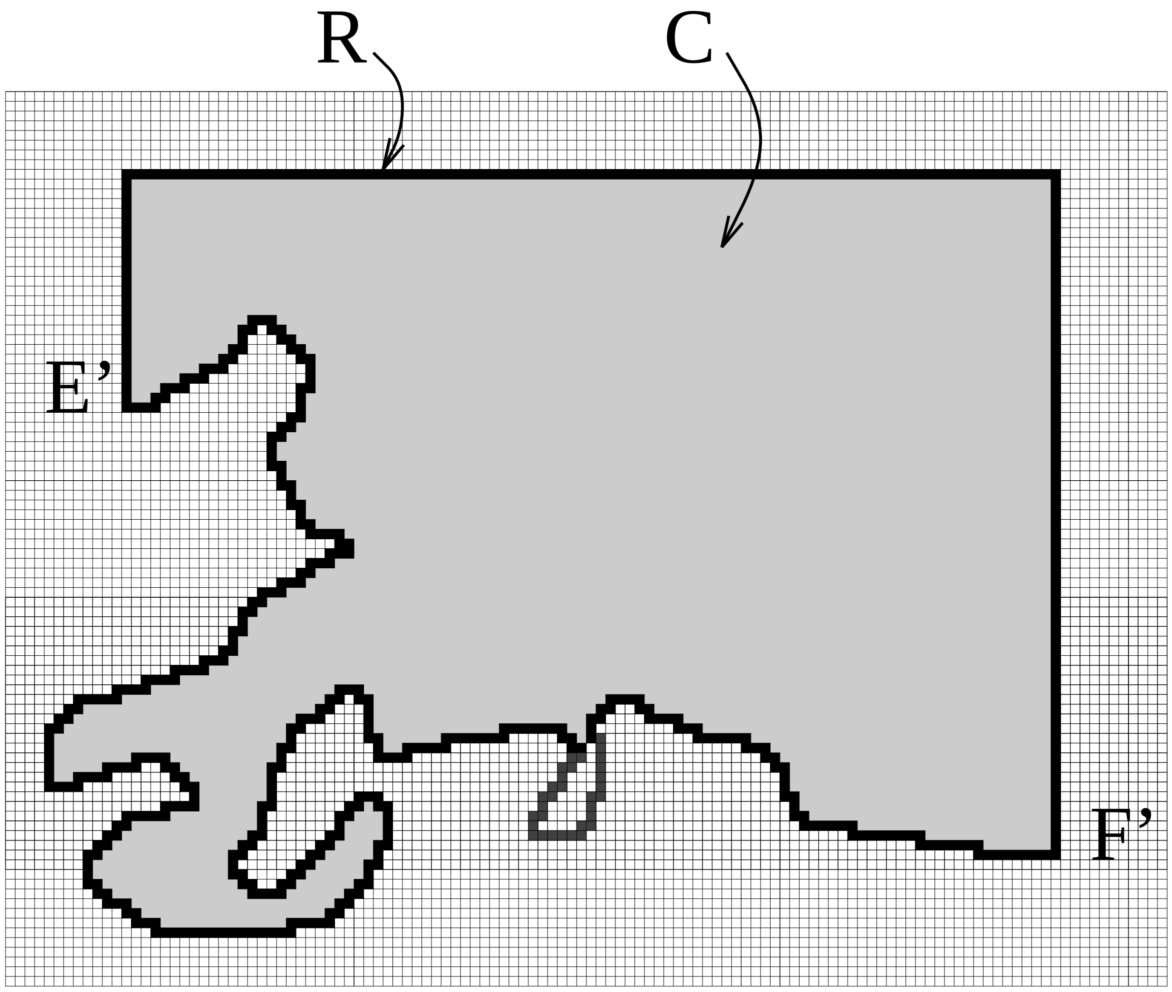

Figure 2.18: Diagonal neighbors path $B_y(t_1)$, connecting E' and F'.

We set $k = \frac{1}{4}n$ and use $l = g(1) + g(3) = O(1)$. The path π_2 must connect each pair x_j, x'_j and x'_j, x_{j+1} within $g(1)$ and $g(3)$, respectively, since π_1 connects them within $\frac{1}{2}|I'| = \Omega(\sqrt{n})$, which is much more than $g(1)$ and $g(3)$. Therefore, π_2 must connect each x_j to x_{j+1} within $g(1)+g(3)$ steps, and traveling from x_0 to x_q (through every x_j, as in figure 2.15), π_2 cannot visit y in this section. Therefore, the distance between x and y on π_2 is at least $\frac{1}{4}|I'| = \Omega(\sqrt{n})$. Clearly, the distance between x and y on π_1 is also at least $\frac{1}{4}|I'| = \Omega(\sqrt{n})$.

However, $d_2(x, y) = l$ and thus x and y must be connected by some path within $g(l) = O(1) < \Omega(\sqrt{n}) \le \frac{1}{4}|I'|$. Contradiction. □

2.4.6 Enhancing the Lower Bound

The proof method of Lemma 9, can be extended to the case where permutations are not paths, but rather a "bounded jumps" path, in which successive elements in the path are not necessarily neighbors but might be at some distance from each other.

Recall that we defined a λ-bounded jumps permutation as one whose jumps on the mesh are bounded by λ. For example, a Hamiltonian path is a λ-bounded jumps permutation with $\lambda = 1$, and a diagonal neighbors path is, in particular, a 2-bounded jumps permutation.

Lemma 11 *Let T_n be the $n \times n$ torus (or mesh), and π_1, π_2 be λ_n-bounded jumps permutations on T_n. For any expansion $g(d) \ge d$ satisfying ${\lambda_n}^3 \left(g(3\lambda_n \cdot g(3\lambda_n))\right)^2 = o(n)$ and large enough n,*

$$\exists x, y \qquad d_2(x,y) \le 3\lambda_n \cdot g(3\lambda_n) \quad \textit{and} \quad \min_{i=1,2}\{d_1(\pi_i(x), \pi_i(y))\} > g(d_2(x,y))$$

where d_1 denotes distance on the (one dimensional) cycle (or line), and d_2 denotes Manhattan distance on the (two dimensional) torus (or mesh).

Proof. Assume n is large enough, such that $\sqrt[4]{n} << \sqrt{n}$ and so on.

As in the proof of Lemma 9, we deal only with respectively small objects (paths, segments etc.), up to size $\frac{1}{2}n$, so we can view our torus as a mesh, and assume w.l.o.g. that the construction takes place in the center of the mesh.

Let s_1 be a segment of π_1 of length $k = \frac{1}{4\lambda_n}n$ (located in the center of the mesh), whose endpoints are E, F (i.e. take a node in the center and a segment of size $\frac{1}{8\lambda_n}n$ to each direction). W.l.o.g. the projection of π_1 on the x-axis (horizontal) is larger than on the y-axis. Denote this projected interval by I, thus, $|I| \ge \sqrt{k}$. Let I_E, I_F be $\frac{1}{10}\sqrt{k}$ intervals around the projection of E and F, respectively. Removing I_E and I_F from I, we get at most 3 sub-intervals of total size $|I \setminus (I_E \cup I_F)| > \frac{1}{2}|I|$, so at least one sub-interval I' is of size $|I'| > \frac{1}{3}(\frac{1}{2}|I|) \ge \frac{1}{6}\sqrt{k}$. Denote this interval as $I' = [c, d]$.

We adjust each of c and d by at most λ_n to get an adjusted interval $I'' = [a, b]$, as follows. Let E' and F' be the highest s_1 nodes in the λ_n-neighborhoods of c and d, respectively. Then a and b would be the x-axis projection of E' and F', respectively.

So $|I''| \geq |I'| - 2\lambda_n > \frac{1}{7}\sqrt{k} = \frac{1}{14}\sqrt{\frac{n}{\lambda_n}}$. Figure 2.19 illustrates the choice of E' and F', (similar to figure 2.16 but with adjustment of $I' = [c, d]$ to get $I'' = [a, b]$). E' and F' define a sub-segment of s_1, denote it by t_1. Let y be a node in the central column of I'', above t_1 in this column and whose distance from t_1 is $l = 2\lambda_n \cdot g(3\lambda_n)$. Trivially, such y must exist and let x be its closest node on t_1, so $d_2(x, y) = l$.

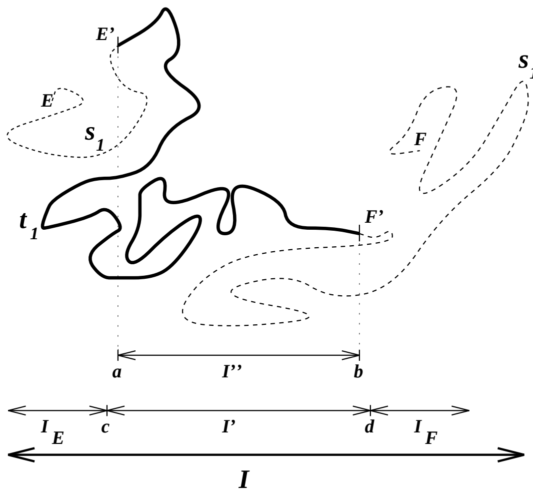

Figure 2.19: Removing endpoints intervals I_E, I_F from s_1, we get t_1 and $I'' = [a, b]$.

We now want to find the boundary of t_1 w.r.t. y (from above). In order to do so we define a border line R starting at E' and going upwards vertically for $2\lambda_n k$ nodes, then horizontally right (or left) until the column of F', and then vertically down to F'. Let $C = C_y(t_1)$ be all nodes reachable from y without using nodes from the border line R nor $N_{\lambda_n}(t_1)$, a neighborhood of t_1, defined as a $\lambda_n \times \lambda_n$ square around each node in t_1, (see figure 2.20).

Let $B = B_y(t_1)$ be all nodes in t_1 in which some node of their λ_n-neighborhood is adjacent to a node in C. We claim that B contains a $2\lambda_n$-bounded jumps path connecting E' and F'. Indeed, if we consider the nodes as squares in a continuous plane, then C is actually a connected set of these squares (add the squares one by one), whose boundary is thus a closed polygonal line. B is all the nodes in t_1, whose λ_n-neighborhood is adjacent to the polygonal line. Going along this closed polygonal line, nodes in B are at most $2\lambda_n$ far from each other (see figure 2.21). In particular, this line connects E' and F'.

We now make sure that $x \in B$, as follows. Go along a shortest path from y to x, and let z be the first node not in C. Node z cannot be in R because the boundary R is much further than l from y, namely, $l << \Omega(\sqrt{k}) \leq \frac{1}{2}|I''|$. So z must be in the λ_n-neighborhood of some $x' \in t_1$. If $x' \neq x$, we take x' instead of x. It is easy to check that changing x with x' is harmless since $d_2(x', x) \leq \lambda_n << \frac{1}{2}l$ so, for example,

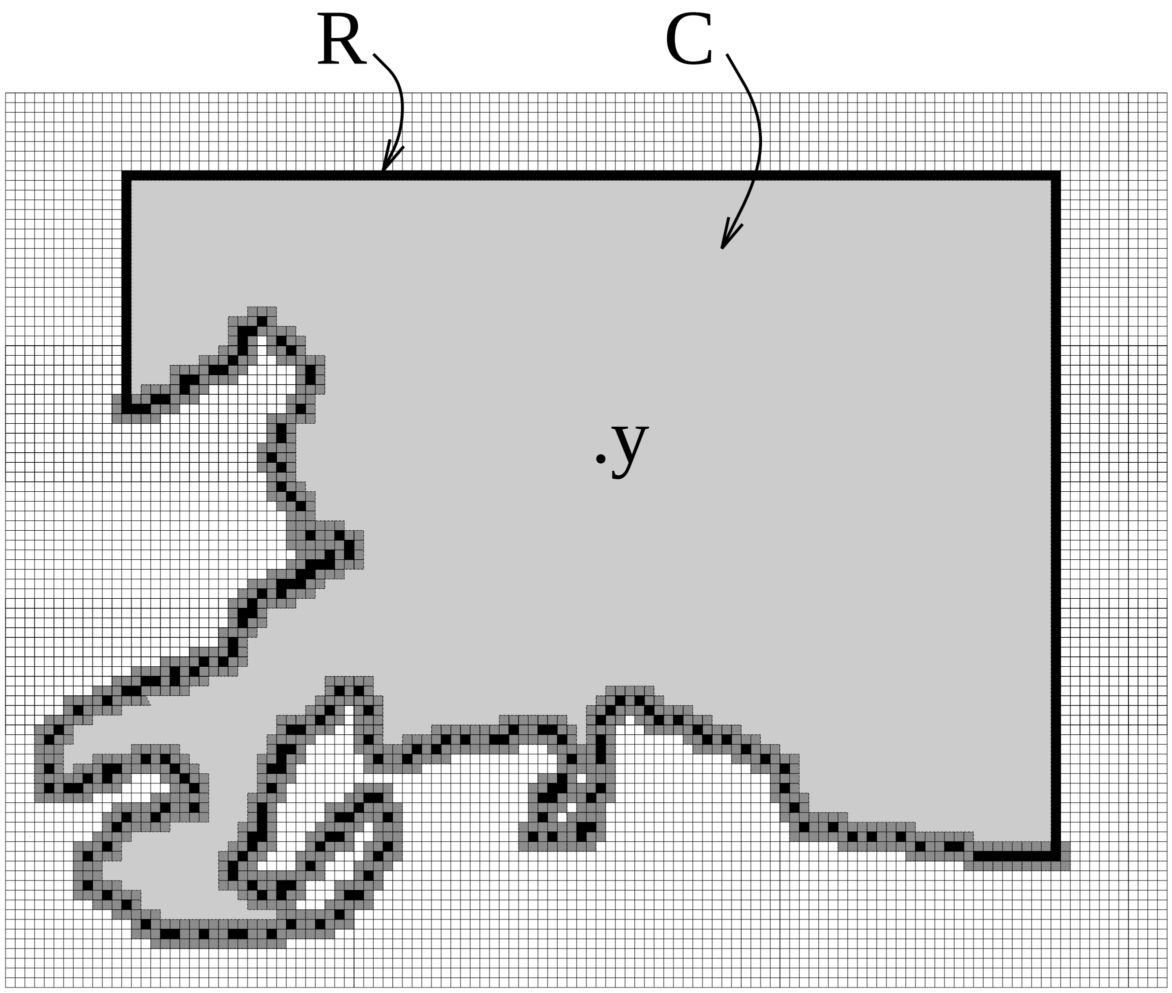

Figure 2.20: Boundary B of t_1 w.r.t. y and neighborhoods of $\lambda_n = 3$.

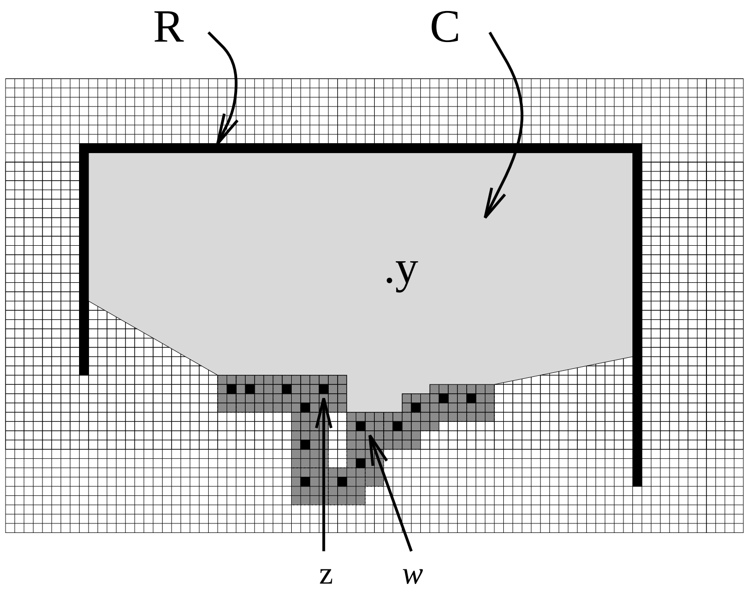

Figure 2.21: Going along the boundary of C (closed polygonal line), the nodes from t_1 are at most $2\lambda_n$ far from each other (in this example $\lambda = 4$). Note that z, w is the worst case, in which $d_2(z, w) = 2\lambda_n$.

$d_2(x', y) > \lambda_n \cdot g(3\lambda_n)$. Now x has a node z in its λ_n-neighborhood which is adjacent to a node in C.

Consider a tour along B, going from E' to F' with jumps of no more than $2\lambda_n$ and visiting x on the way. Denote the central segment of this tour (spreading $\frac{1}{4}|I''|$ wide for each side of x) as $x_0, x_1, x_2, \ldots, x_r = x, \ldots, x_q$. It is a $2\lambda_n$-bounded jumps path, so for all j, $d_2(x_j, x_{j+1}) \leq 2\lambda_n$. By definition of B, any x_j has a λ_n-neighbor $x'_j \in C$, which is not on t_1.

Obviously, $d_2(x'_j, x_{j+1}) \leq 2\lambda_n + \lambda_n \leq 3\lambda_n$, so they must be connected by some π_i within $g(3\lambda_n)$ steps. But x'_j is not on s_1, so their distance on π_1 is at least $\frac{1}{\lambda_n}\frac{1}{4}|I''| \geq \Omega(\sqrt{n/\lambda_n^3}) >> g(3\lambda_n)$ step (we took only the central segment of I'''). Therefore, their distance on π_2 must be at most $g(3\lambda_n)$. The same argument applies also to x_j and x'_j, which are even λ_n-neighbors and therefore must be connected by π_2 with distance $g(3\lambda_n)$. Combine the two results (recall an expansion function is monotonic),

$$d_1(\pi_2(x_j), \pi_2(x_{j+1})) \leq 2g(3\lambda_n)$$

Consider now π_2 visiting x_0. It must get to x_1 within $2g(3\lambda_n)$ steps, to x_2 within the following $2g(3\lambda_n)$ steps and so on, going through $x = x_r$ and until reaching x_q. Therefore, the maximal distance it can get far from the border in this section (from x_0 to x_q) is $\frac{1}{2}\lambda_n \cdot 2g(3\lambda_n) < l$. So the minimal distance from x to y on π_2 is

$\frac{1}{\lambda_n}(\frac{1}{4}|I''| - \lambda_n) \geq \frac{1}{5\lambda_n}|I''|$ (recall y is close to the central column of I'').

The distance of x and y on π_1 is also at least $\frac{1}{\lambda_n}\frac{1}{2}(|I''| - \lambda_n) > \frac{1}{5\lambda_n}|I''|$. However, our assumption requires that some path connects x, y within

$$g(3\lambda_n \cdot g(3\lambda_n)) << \Omega(\sqrt{n/\lambda_n^3}) \leq \frac{1}{5\lambda_n}|I''|$$

steps. Contradiction.

□

The result of Lemma 11 is that either we use large jumps and then some distance shrinks considerably, or some distance expand by more than squaring. But for permutation families $\mathcal{F}$ of cardinality two, $d_{line} = \Theta(d_{mesh}^2)$ is impossible.

2.5 Upper Bound for Higher Dimensional Torus

Observation 12 *Let $\mathcal{F}$ be a stereoscopic family of permutations on the m-dimensional mesh (or torus). If $|\mathcal{F}| = O(1)$ then $g_N(d) = \Omega(d^m)$.*

Proof. Consider the following counting argument. Let x be some fixed node on the m-dimensional mesh (or torus). Then the number of nodes on the mesh (or torus) in radius R from x is $\Theta(R^m)$. Since the number of paths in $\mathcal{F}$ is constant, at least one of these nodes must be $\Omega(R^m)$ steps far from x on every single path. Therefore $g_N(d) = \Omega(d^m)$. □

Definition 11 *A* **Gray code** *of $\{0,1\}^m$, denoted G_m, is a permutation of $\{0,1\}^m$, such that the Hamming distance (L^1 norm) between any two successive elements is 1.*

Let G_m be a Gray code on $\{0,1\}^m$, say from $(0,0,\ldots,0)$ to $(1,0,0,\ldots,0)$. We shall denote it by: $(0,0,\ldots,0) \xrightarrow{G_m} (1,0,\ldots,0)$. Clearly, G_m^{-1} (same code in reverse order) is also a Gray code, and it will be denoted as: $(1,0,\ldots,0) \xrightarrow{G_m^{-1}} (0,0,\ldots,0)$.

Lemma 13 *Let T_n be an m-dimensional torus ($n \times n \times \cdots \times n$), for $n = s \cdot 2^l$, arbitrary $l > 0$ and odd $s \in \{m+1, m+2\}$. Then there exists a stereoscopic family of permutations $\mathcal{F}^{(m)} = \{\pi_1, \pi_2, \ldots, \pi_{m+1}\}$ of Hamiltonian cycles on T_n, such that*

$$\forall x, y \qquad \min_{i=1,2,\ldots,m+1}\{d_1(\pi_i(x), \pi_i(y))\} \leq (2(m+2)d_m(x,y))^m$$

where d_1 denotes distance on the (one dimensional) cycle, and d_m denotes Manhattan distance on the (m-dimensional) torus.

Proof. Denote each torus node as a tuple $(x_1, x_2, \ldots, x_m)$ where $1 \leq x_1, x_2, \ldots, x_m \leq n$. We generalize the two-dimensional proof of Lemma 6 and show a specific Hamiltonian cycle π. In fact, we construct a Hamiltonian path from $(1, 1, \ldots, 1)$ to $(n, 1, 1, \ldots, 1)$ on the corresponding mesh M_n, giving rise to a cycle on the torus.

The Hamiltonian path π on M_n is constructed recursively, as follows. At each phase we partition M_n to 2 segments in each axis, producing 2^m equal size sub-meshes. Alternatively, one can view the partitioning to be a high-level description of M_n as a cube broken into 2^m sub-cubes. A hyperplane separating adjacent sub-cubes of size $d \times d \times \cdots \times d$ in this description will be called a ***d*-facet**. The recursion stops when reaching sub-meshes whose size is $s \times s \times \cdots \times s$.

At each phase, we are also given entry and exit nodes (placed in adjacent corners in the mesh), and we compose accordingly a simple path from the entry node to the exit node, traversing each sub-mesh exactly once. We only select entrance and exit nodes for each sub-mesh (with the requirement that they reside in adjacent corners), and the exact traversal in the each sub-mesh is constructed by successive phases of recursion.

We have to show that the path defined at each phase of the recursion is indeed feasible. Namely, given entrance and exit nodes at adjacent corners of the mesh, one can always define a traversal order on the sub-meshes. Let us denote the partitioning to sub-meshes as $\{0, 1\}^m$, where each sub-mesh is named as a binary m-tuple. W.l.o.g. the entrance node is the corner of sub-mesh $(0, 0, \ldots, 0)$ and the exit is a corner of sub-mesh $(0, 0, \ldots, 0, 1)$. Then the following recursive definition of Gray code, is a feasible path:

$$G_m : (0, \ldots, 0; 0) \overset{G_{m-1} \times \{0\}}{\longrightarrow} (1, 0, \ldots, 0; 0) \rightarrow (1, 0, \ldots, 0; 1) \overset{(G_{m-1})^{-1} \times \{1\}}{\longrightarrow} (0, \ldots, 0; 1)$$

The adjacent corners requirement is easy to satisfy, since every adjacent sub-mesh can be reached through some adjacent corner.

Define π_i $(i = 0, \ldots, m)$ to be a $\frac{i \cdot n}{s}$ units shift of π in each axis (coordinate) on the torus T_n (a linear transformation modulo n).

Let x, y be arbitrary nodes on the torus, and we will bound their distance on the π_i's. Denote $d' = d_m(x, y)$ their distance on the m-dimensional torus. Let $d = s \cdot 2^j$ be the smallest possible such that $d > s \cdot d'$, so clearly $d \leq 2sd'$. Consider the partitioning of the torus into sub-meshes of size $d \times d \times \cdots \times d$ (i.e. after $\log_2 \frac{n}{d} = l - j$ phases of the recursive construction).

Observe that no two d-facets of different paths are closer than $\frac{d}{s}$. Note that partitioning of n to $\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \ldots$ does not coincide with the shift by $\frac{1}{s}$, and n is chosen to be divisible by all these denominators. Formally, assume w.l.o.g. that f_1 is a d-facet of π_1 located at indices which are multiples of d, and f_2 is a d-facet of π_i located at indices which are $\frac{in}{s}$ plus multiples of d. Consider the location of π_i relatively to f_1, i.e. its location modulo d (since f_1 appears at multiples of d). It comes down to resolve $\frac{in}{s}$ modulo d, and $\frac{in}{s} = i2^l = d\frac{i2^{l-j}}{s}$. Since $i2^{l-j}$ is not a multiple of s, we conclude that the difference between adjacent f_1 and f_2 is some multiple of $\frac{d}{s}$.

Finally, we claim that x and y belong to the same $d \times d \times \cdots \times d$ sub-mesh in at least one cycle π_i. Hence their distance on this π_i is at most the size of the sub-mesh, and we're done:

$$d_1(\pi_i(x), \pi_i(y)) < d^m \leq (2sd')^m \leq (2(m+2)d_m(x,y))^m$$

Otherwise, x and y are in different $d \times d \times \cdots \times d$ sub-meshes in all Hamiltonian cycles. In other words, x, y are separated from each other by at least one facet in any π_i $(i = 0, \ldots, m)$. But since there are only m possible facet orientations, at least two must be of the same orientation. However the minimal distance between any two such facets is $\frac{d}{s}$, and therefore $d_m(x,y) \geq \frac{d}{s} > d' = d_m(x,y)$. Contradiction. $\square$

The paths used in this construction also have a complementary property. When the nodes of the line are placed on the mesh along the path, their distances on the line shrink to their m-th root.

Lemma 14 *Let π be the path constructed recursively in the proof of Lemma 13. Then π is a shrinkable numbering of vertices with $\alpha = \frac{1}{m}$, and $\beta = 2(m+1)$. That is*

$$\forall x, y \qquad d_m(x,y) \leq 2(m+1)\sqrt[m]{d_1(\pi(x), \pi(y))}$$

Proof. Denote $d = d_1(\pi(x), \pi(y))$. Then there is some $d' \times d' \times \cdots \times d'$ square in the construction such that $\sqrt[m]{d} \leq d' < 2\sqrt[m]{d}$. Assume W.l.o.g. that x appears before y on the path. Then traveling on the path from x to y (total of $d-1$ steps) one can either stay in the same $d' \times d' \times \cdots \times d'$ square, or advance to a neighboring one, but not further (because the square's size is $(d')^m \geq d$). Either case, on the mesh x and y share a rectangle with $m-1$ sides of length d' and one of $2d'$, thus

$$d_m(x,y) \leq (m-1)d' + 2d' \leq 2(m+1)\sqrt[m]{d}$$

$\square$

2.6 Stereoscopic Families for Arbitrary Dimensions

In this section we finally relate to the most general notion of stereoscopic families of permutations, used to map an m-dimensional universe to a q-dimensional one. So far we discussed only the case where $q = 1$, although our definition refers to the general form, allowing any one-to-one functions (see Definition 8).

We present an explicit construction, based on previous sections. The construction is basically a composition of mapping to the one-dimensional universe (supported by Lemma 13), and then mapping to the q-dimensional universe by a shrinkable numbering of vertices.

On the one hand, the mapping to one-dimension requires $m+1$ permutations, and a direct mapping could possibly do better. Consider the degenerate case where $m = q$, and π_1^{-1} is composed on each of $\pi_1, \ldots, \pi_{m+1}$. The composition of π_1^{-1} on π_1, yields the identity function, which suffices by itself. The point is that although some pairs of vertices were not handled by π_1, and their distance on the one dimensional universe might be very large, π_1^{-1} shrinks their distance back so good, that the remaining permutations $\pi_2, \ldots, \pi_{m+1}$ are not needed. This phenomena might indicate that also in the general case, some of the permutations are redundant.

On the other hand, passing through the one-dimensional universe can be very useful, as in the problem of hashing noisy data (see section 2.8).

Observation 15 *Let $\mathcal{F}$ be a stereoscopic family of permutations from the m-dimensional mesh (or torus) to the q-dimensional mesh (or torus). If $|\mathcal{F}| = O(1)$ then $g_N(d) = \Omega(d^{\frac{m}{q}})$.*

Proof. Consider the following counting argument. Let x be some fixed node on the m-dimensional mesh (or torus). Then the number of nodes on this mesh (or torus) in radius R from x is $\Theta(R^m)$. Since the number of functions in $\mathcal{F}$ is constant, at least one of these nodes must be $\Omega(R^{\frac{m}{q}})$ steps far from x in the image of every single function of $\mathcal{F}$. Therefore $g_N(d) = \Omega(d^{\frac{m}{q}})$. $\Box$

Lemma 16 *Let $T_n^{(m)}$ be an m-dimensional torus ($n \times n \times \cdots \times n$), for $n = s \cdot 2^l$, arbitrary $l > 0$ and odd $s \in \{m+1, m+2\}$. Let $T_{n'}^{(q)}$ be a q-dimensional torus ($n' \times n' \times \cdots \times n'$), such that $|T_n^{(m)}| = n^m = (n')^q = |T_{n'}^{(q)}|$. Then there exists a stereoscopic family of permutations $\mathcal{F}_q^{(m)} = \{\pi_1, \pi_2, \ldots, \pi_{m+1}\}$ of permutations $\pi_i : T_n^{(m)} \mapsto T_{n'}^{(q)}$, such that*

$$\forall x, y \qquad \min_{i=1,2,\ldots,m+1} \{d_q(\pi_i(x), \pi_i(y))\} \leq O\left((d_m(x,y))^{\frac{m}{q}}\right)$$

where d_q denotes Manhattan distance on the q-dimensional torus $T_{n'}^{(q)}$, and d_m denotes Manhattan distance on the m-dimensional torus $T_n^{(m)}$.

Proof. Let $\mathcal{G}^{(m)} = \{\tau_1, \tau_2, \ldots, \tau_{m+1}\}$ be the stereoscopic family of Hamiltonian paths guaranteed by Lemma 13. Let σ^{-1} be a shrinkable numbering of vertices (by Lemma 14, for example) with $\alpha = \frac{1}{q}$ and some constant β, on the q-dimensional torus $T_{n'}^{(q)}$.

Define $\pi_i = \sigma \circ \tau_i$. Then $\mathcal{F}_q^{(m)} = \{\sigma \circ \tau_1, \ldots, \sigma \circ \tau_{m+1}\}$ is the requested stereoscopic family. To see this, let x and y be arbitrary nodes in the m-dimensional torus. By Lemma 13,

$$\exists i,\ d_1(\tau_i(x), \tau_i(y)) \leq (2(m+2)d_m(x,y))^m$$

By definition of σ,

$$\forall i, \qquad d_q(\sigma(\tau_i(x)), \sigma(\tau_i(y))) \leq \beta(d_1(\tau_i(x), \tau_i(y))^{\frac{1}{q}}$$

So we're done:

$$\exists i, \qquad d_q((\sigma \circ \tau_i)(x), (\sigma \circ \tau_i)(y)) \leq \beta(2(m+2)d_m(x,y))^{\frac{m}{q}}$$

$\square$

2.7 Routing along Stereoscopic Families

We relate to hot-potato routing paradigm on the two dimensional mesh (or torus), where each link (edge) has capacity of 3 channels, i.e. each link can transfer 3 packets in a single time step. Such networks can be implemented by multiplexing 3 packets on each link.

In this network model, packets travel only along the 3 Hamiltonian paths of a corresponding stereoscopic family of permutations, $\mathcal{F} = \{\pi_1, \pi_2, \pi_3\}$. Since the capacity of each link is 3, packets traveling along different paths do not interfere with each other. Obviously, each packet prefers the path on which its distance to destination is the shortest among $\mathcal{F}$. So at the time of injection, every source node assigns its packet to this preferred path, and the packet will travel along this path until reaching its destination.

Consider batch routing scenario where each node is the source of at most one packet. Then each packet will be injected to the network on its preferred path $\pi_i \in \mathcal{F}$, and travel along this path. The routing is collision-free since the 3 channels enable the coexistence of the 3 paths, without any contentions. According to Lemma 13, each packet will reach its destination after $36{d_p}^2$ steps and with no deflections.

The batch routing can be easily extended to dynamic routing by allowing packets injection, on a vacancy basis. A node generating a new packet, finds the preferred (best) path according to its destination, and waits for an opportunity to inject the packet to this path. From the moment the packet is injected, it travels with no further delays, reaching its destination within the next $\Theta({d_p}^2)$ steps.

The main advantage of routing along stereoscopic families is its simplicity. Initially, the injecting node assigns a path to the packet (possibly by a fixed table prepared in advance). At intermediate nodes, the assignment labor of packets to outgoing edges is almost trivial. If the packet is destined to the current node, no routing needs to be done. Otherwise, the incoming edge alone defines the assignment to an outgoing edge, according to predefined decisions (which reflect the Hamiltonian paths). There is no need to consider the destination of the packet, nor make any calculations. The trivial routing table requires minimal computational resources (both time and memory).

2.8 Other Applications (Hashing)

A **dictionary** is a data structure for storing elements from universe U in memory M. It has to be capable of storing any subset $D \subset U$ whose cardinality is not too big, and provide efficient implementation of the following operations:

- Queries: Membership (Given $x \in U$ decide whether $x \in D$) and Find (determine where x is stored in M).
- Update operations: Add/Delete/Change elements in D.

We relate to the "noisy" version of the problem, suggested by Linial and Sasson [20], where inputs might have uncertainty, allowing several instances of the same input to be similar (rather than identical). For completeness, we repeat their definition. Let U be a metric space with distance function d_U, reflecting the structure of the inputs. Then the dictionary has to provide the following operations:

- Queries: Membership (Given $x \in U$ and $\Delta > 0$ (noise/uncertainty), list all items $y \in D$ which are close to x, namely $d_U(x, y) \le \Delta$); and Find (determine where are these items stored in M).
- Update operations: Add/Delete/Change elements in D.

Any dictionary can be adapted to deal with noisy data as well. Given $x \in U$, check for every y in the Δ-neighborhood of x (denoted by Δ_x), whether it is stored in the dictionary. However, this procedure has two flaws:

1. Each check of $y \in \Delta_x$ requires an application of the hashing function. Overall, the procedure may require $\Omega(|\Delta_x|)$ applications of the hashing function.
2. Elements $y \in \Delta_x$ might be hashed to distant locations in the memory. Large memory is usually paged, in which case the procedure might require access to $\Omega(|\Delta_x|)$ different pages.

The problem of hashing noisy data appears in several variants in many applications. Minsky and Papert [22] define the *Best Match* problem, where D is a dictionary of strings over some alphabet Σ, and it is required to store D in such as way that on query q, the member of D closest to q in Hamming distance be retrieved. The version where all members in D of distance d or less from q need to be retrieved is the *Approximate Query (AQ)* problem of [14]. Most of the work on these problems (see [24] and the references therein) focuses on deterministic schemes.

Linial and Sasson [20], devise a "noisy" hashing scheme for a one dimensional universe U and a one dimensional memory M, with any noise measure Δ. It is a non-expansive hashing scheme, i.e. one which translates every Δ-neighborhood in U to a constant number of Δ-neighborhoods in M. Linial and Sasson define a specific family $\mathcal{H}$ of functions with the property that for every $x \in U$, either $f(x+1) = f(x) + 1$

or $f(x+1) = f(x) - 1$. Such function is a long path with "turning points" on the interval U, and can be specified by its "starting point" and its "turning points". They restrict $\mathcal{H}$ to functions f whose "turning points" are selectively chosen. Hashing the dictionary D then uses several hashing tables, each corresponding to a different hash function $f \in \mathcal{H}$.
We introduce the following notation:

$H_{1,1}$ hashing scheme for the one dimensional universe and one dimensional memory.

$H_{d,q}$ hashing scheme for d-dimensional universe and q-dimensional memory, with Manhattan distance (L^1 norm).

$C_{d,q}^{Member}(N, \Delta)$ number of pages (blocks of size $|\Delta_x|$) accessed by a Membership query of the $H_{d,q}$ scheme.

$C_{d,q}^{Update}(N)$ number of pages accessed by an Update operation (such as Add, Delete and Change) of the $H_{d,q}$ scheme.

$|M_{d,q}|$ memory complexity of the $H_{d,q}$ hashing scheme.

The non-expansive hashing scheme [20] is a $H_{1,1}$ scheme. For arbitrary $\epsilon > 0$, it uses $O(\log \frac{1}{\epsilon})$ hashing tables, each of size $|D|^{\frac{1}{1-\epsilon}}$, with the following complexity:

$$C_{1,1}^{Member}(N, \Delta) = O(\log \frac{1}{\epsilon})$$

$$C_{1,1}^{Update}(N) = O(\log \frac{1}{\epsilon})$$

$$|M_{1,1}| = O(\log \frac{1}{\epsilon} \cdot |D|^{\frac{1}{1-\epsilon}})$$

Our results on stereoscopic families of permutations combined with any dictionary scheme for the one dimensional universe create a dictionary for universe U and memory M of arbitrary dimensions, each with Manhattan distance (L^1 norm). In particular, we extend the non-expansive hashing scheme devised by Linial and Sasson (denoted here as $H_{1,1}$). In the spirit of Lemma 16, we show how to construct a scheme for d-dimensional universe with q-dimensional memory ($H_{d,q}$).

The general scheme, $H_{d,q}$, is based on a stereoscopic family of permutations on the d-dimensional torus, and a shrinkable numbering of vertices in the q-dimensional mesh (or torus). Lemma 13 guarantees the existence of such a family $\mathcal{F} = \{\pi_1, \ldots, \pi_{d+1}\}$ of cycles, which are, in particular, permutations. Lemma 14 shows how to construct shrinkable numbering of vertices.

The dictionary D is first mapped to $d+1$ separate images on the (one dimensional) interval, one for each cycle π_i ($i = 1, \ldots, d+1$). Each of these $d+1$ intervals is then hashed by the non-expansive hashing $H_{1,1}$ to another one dimensional representation. Finally, the one dimensional tables are, in turn, mapped by the shrinkable numbering of vertices to the q-dimensional mesh (or torus).

Membership query is straightforward. Given an input $x \in U$, find its $d+1$ images in the q-dimensional memory, and check their corresponding neighborhoods. The stereoscopic family of permutations $\mathcal{F}$ enlarges distances (the noise) polynomially to $O(\Delta^d)$. In the non-expansive hashing distances do not expand, and the shrinkable numbering shrinks distances polynomially to $O(\Delta^{d/q})$. So,

$$C_{d,q}^{Member}(N, \Delta) = (d+1) \cdot C_{1,1}^{Member}(N, O(\Delta^{d/q}))$$

Update operations (such as Add, Delete and Change) require a corresponding update in each of the $d+1$ segments, therefore

$$C_{d,q}^{Update} = (d+1) \cdot C_{1,1}^{Update}$$

The stereoscopic family of permutations $\mathcal{F}$ copies $d+1$ times each dictionary element (or its index or a pointer to the element). The shrinkable numbering of vertices requires the same size of memory as its domain. Hence, the $H_{d,q}$ scheme requires memory complexity of $|M_{d,q}| = (d+1)|M_{1,1}|$.

2.9 Conclusion and Open Problems

We presented the concept of stereoscopic families of permutations and its relevance to hot-potato routing, and demonstrated an additional application of this concept. We saw a construction for the two dimensional mesh and torus, and proved a matching lower bound for stereoscopic family of 3 Hamiltonian paths. Nevertheless, several related problems remain open.

- *Unbounded Permutations.*

  We showed a lower bound for λ_n-bounded jumps permutations, with $\lambda_n = o(\sqrt[11]{n})$. What is the minimal possible cardinality of stereoscopic families, if the jumps are unbounded (arbitrary permutations) ?

- *One Sequence with Several Appearances.*

  Assume that the t independent permutations are incorporated into one long sequence, such that each element appears t times in the sequence. What would be the minimal t needed to bound expansion to quadratic order ?

- *Undirected Paths.*

  Our interpretation of routing by stereoscopic families of permutations uses 6 directed paths to implement the 3 undirected ones, due to the directed nature of routing. How many directed paths (or permutations) are needed to attain quadratic expansion ?

The two dimensional construction was generalized to a stereoscopic family with $m+1$ Hamiltonian cycles on an arbitrary m-dimensional universe. This was used to present a specific construction for the general case, where an m-dimensional universe is mapped to a q-dimensional one through an intermediate one dimensional universe. The mapping is achieved by a stereoscopic family of $m+1$ permutations. In both situations, we reach the optimal expansion g, but not necessarily the optimal cardinality of the stereoscopic family. Thus, important questions regarding the general case remain open.

- *Higher Dimension Lower Bound.*

  Is it possible to find a stereoscopic family of permutations for an m-dimensional universe with less than $m+1$ permutations ?

- *Direct Mapping between Arbitrary Dimensions.*

  We indicated that a direct mapping between arbitrary dimensions might use fewer permutations than our indirect construction.

Chapter 3

Minimum Advance Algorithms on Trees

In this chapter we discuss minimum advance algorithms on tree networks. Our work in this area focuses on the minimal requirement needed to guarantee livelock avoidance on tree networks.

We show that the minimum advance principle suffices to ensure evacuation of any routing problem on any tree networks. However, we also show an example where this principle is not enough to guarantee $t_p \leq d_p + 2(k-1)$.

Note that the high-end class of maximum advance algorithms is known to achieve $t_p \leq d_p + 2(k-1)$ on trees (see [11, 8]). The question whether the mid-range classes, namely weakly stable and stable algorithms, achieve this bound remains open.

Definition 12 *A* **configuration** *of packets in a network is any placement (mapping) of the packets to the nodes of the network.*

We denote by Γ the set of all possible configurations of packets in the network. So for k packets on a network with n nodes, $|\Gamma| \leq n^k$.

Lemma 17 *Any minimum advance routing algorithm delivers at least one packet to its destination (without livelock).*

Proof. Let T be a tree network consisting of $n > 0$ nodes, and $k > 0$ packets, and let $\sigma = c_1, c_2, c_3, \ldots$ be a sequence of configurations ($c_i \in \Gamma$) describing the execution of a minimum advance routing. Let C be the set of all configurations participating in the sequence σ, i.e. $C = \{c_1, c_2, \ldots\}$.
Proceed by induction on $|C|$.

(i) If $|C| = 1$ then the network does not change, and this is impossible.

(ii) We assume that the sequence cannot livelock for any $C' \subset C$, and show it holds also for C. Consider c_1, the first configuration in the sequence. If it does not appear anywhere else in the sequence, we can use the induction hypothesis with $\sigma' = c_2, c_3, \ldots$ and $C' = C \setminus \{c_1\}$, to conclude that some packet is delivered to its destination.

Otherwise, c_1 appears later on in the sequence, say at time j ($c_1 = c_j$). In the first step ($c_1 \mapsto c_2$) there is some advancing packet, say p going from u to v. The edge (u, v) separates T to two subtrees, T_1 (which includes both p's destination and v) and T_1' (which includes u). Hence, p advances from T_1' to T_1, as shown in figure 3.1.

Consider now the next time step ($c_2 \mapsto c_3$).

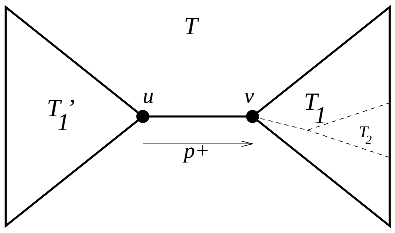

Figure 3.1: Packet p separates the tree

1. If p advances again, then this advance similarly defines T_2, T_2', such that $T_2 \subset T_1$.
2. Otherwise, p is deflected from v, so some other packet, p', must advance from v.
   (a) If p' advances but not on edge (v, u), then it similarly defines T_2, T_2', such that $T_2 \subset T_1$.
   (b) If p' advances on edge (v, u), then p is assigned to some other edge and in T_1. Since configuration c_1 is repeated at time j, p must traverse the edge back from v to u at some time before time j. By the minimum advance principle, if p is deflected from v to u, then some other packet p'' must advance into T_1. Hence p'' similarly defines T_2, T_2', such that $T_2 \subset T_1$.

We repeat this argument, unless we reach a configuration that does not appears later on in the sequence σ. If we do reach such a configuration, we can remove it and use the induction hypothesis with $C' \subset C$, to conclude that some packet is delivered to its destination.

Otherwise, we find $T_2, T_3, \ldots, T_l$, and within n iterations we must reach some $|T_l| = 1$. In this case, we're done because the packet advancing into T_l has just reached its destination.

□

Corollary 18 *Any minimum advance routing algorithm evacuates any tree network within n^k steps.*

Proof. No configuration can repeat itself or a livelock would have been possible. Hence, the routing terminates in at most $|\Gamma| \leq n^k$ steps. □

Observe that the proof of Lemma 17 does not assume any kind of local decisions. An adversary may control the behavior of the algorithm based on global information, as a node is allowed to act inconsistently in case of similar or repeating situations. The only constraint used in the proof is that the execution sequence obeys the minimum advance principle.

Observation 19 *Minimum Advance algorithm on a tree network does not necessarily deliver packets within $d_p + 2(k-1)$ steps.*

Proof. We show a counter example of routing 3 packets (denoted a, b and c) on a small tree network. The routing complies to the Minimum Advance principle, but packet a is deflected 3 times, and thus delivered only after $d_p + 6 > d_p + 2(k-1)$ steps. The exact steps sequence is described in figure 3.2. Circles denote packets destinations and arrows represent the traversal of a packet. □

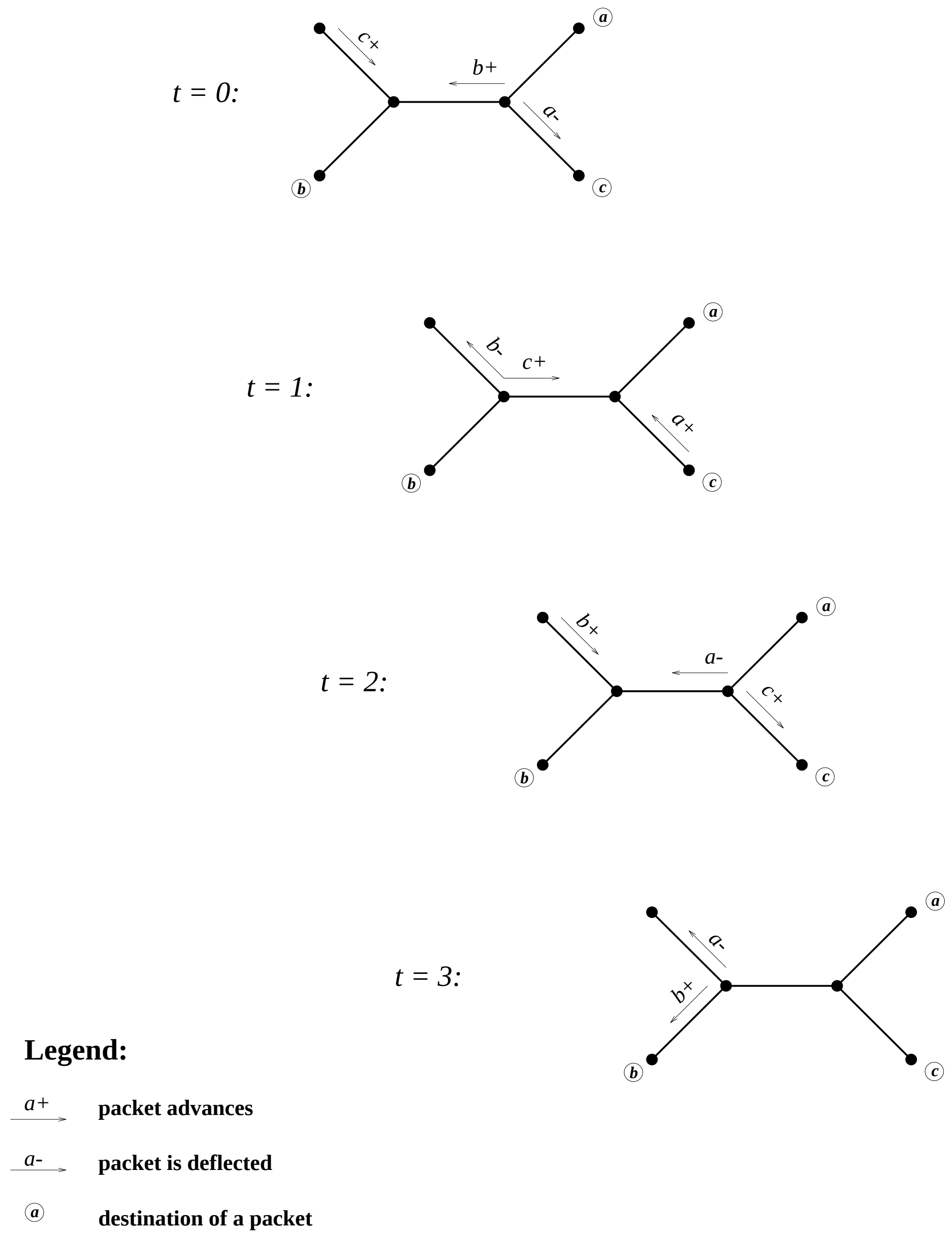

Figure 3.2: An example of minimum advance routing on a tree network, where delivery of packet a takes more than $d_p + 2(k-1)$.

Chapter 4

Potential Guided Routing Algorithms

4.1 Motivation and Main Results

We define a class of simple and natural hot-potato routing algorithms, called *potential guided algorithms*. Each packet p has its "own" potential function, which defines the packet's *potential value* at each of the network nodes $\Phi_p : V \mapsto \mathbb{R}_0^+$. The potential at a node depends on the current location (this node) and the packet's destination. The sole constraint on the potential function Φ_p is that every node, except the destination, has a neighbor with lower potential value.

The routing algorithm in each node tries to minimize the potential locally. The node assigns each incoming packet to an outgoing edge such that its potential will be minimal (if possible under congestion restrictions). Hence, the route selected for a packet is "guided" by the packet's potential function, with deviations occurring due to contentions and deflections.

We consider a whole set of simple and natural algorithms in which the potential depends only on the two basic arguments, the packet's destination and the current location. This also enables to compute in advance for each node a table of potentials for each possible destination and each neighbor (outgoing edge). More sophisticated potential functions might take into account the packet's origin, the current time step and so on, but are beyond the scope of this work.

We also restrict ourselves to a specific contention resolution scheme. Namely, each node assigns the incoming packets to outgoing edges, such that the sum of their potentials in the next step will be the minimum possible (*minimum sum of potentials*). We ignore other possible contention resolution schemes such as fixed priorities, closest first, etc.

The routing performance is highly dependent on the choice of the potential functions. A clever choice of the potential functions may lead to good performance. Our goal is to find out what conditions (properties) are sufficient to ensure good performance. Note, for example, that defining the potential of a packet to be its distance from its destination, results with a maximum advance algorithm.

The following property of potential guided algorithms avoids livelocks on any network. For every packet at every node, the potential (of the packet) at the current node is higher than the average potential (of the packet) at the node's neighbors. When this property holds, even a random assignment is expected to strictly reduce the sum of potentials. Hence, there always must exist some assignment which improves the sum of potentials. At each step of the routing the overall sum of potentials in the network is strictly reduced, and hence the network cannot enter a livelock.

In section 4.5 we show that on tree networks any potential guided algorithm is stable and, in particular, minimum advance. By Chapter 3, such routing cannot livelock. However, potential guided algorithms are not necessarily maximum advance, even not on tree networks.

We suggest a specific potential function which is the expected hitting time of a random walk from the packet's current location to its destination. We refer to this algorithm as the *expected hitting time routing algorithm.* The expected hitting time potential at a node is higher than the average potential at its neighbors, unless this is the destination node. As mentioned before, this property guarantees termination of any problem in any network (for details see section 4.6.1).

On general networks, we show that the expected hitting time routing algorithm evacuates any problem within $2H_{max} \log k$ steps, where H_{max} is the maximum hitting time between any pair of nodes in the graph. We show that the expected hitting time routing algorithm is maximum advance on tree networks, and therefore achieves $t_p \leq d_p + 2(k-1)$. On cycles, this algorithm is maximum advance with closest first priority, and also achieves this bound. The expected hitting time routing algorithm on mesh networks does not use shortest paths, so it definitely cannot achieve $t_p \leq d_p + 2(k-1)$. Our upper bound, based on the general case, is evacuation within $O(n^2 \log n \log k)$ steps.

4.2 Definitions

We introduce terminology which we will use throughout this chapter.

Definition 13 *A* **schedule** *is a mapping S of every packet and every time step to a node in the graph (its location at that time). A Schedule is* **proper** *if it routes all packets to neighbors (packets cannot jump and cannot stay in their place), and always keeps the capacity of each link at most 1.*

Definition 14 *A* **potential function** *for packet p, whose destination is $v_0 \in V$, is a function $\Phi_p : V \mapsto \mathbb{R}_0^+$ such that every node other than the destination v_0 has a neighbor with lower potential. Namely,*

$$\forall v \in V \setminus \{v_0\}, \quad \exists (v,u) \in E, \qquad \Phi_p(u) < \Phi_p(v)$$

Definition 15 *The* **potential of an assignment** *A at a node v, denoted by $\Phi_v(A)$, is the sum of potentials of the packets in v after applying assignment A (i.e. at the next time step).*

Definition 16 *Assume that every packet in some network has a corresponding potential function. A* **potential guided algorithm** *is one which satisfies the following rule at every time step. Each node chooses the assignment with the minimum potential. That is, incoming packets are assigned to outgoing edges such that the sum of the packets' potentials in the next time step (after traversing their edges), will be minimum.*

Definition 17 *A* **random walk** *on a finite graph is an infinite path, where at each step the random walk moves to a vertex chosen at random with uniform probability from the neighbors of the current vertex.*

Definition 18 *The* **expected hitting time** *from node u to node v, denoted by $H(u, v)$ is the expected number of steps in a random walk starting at u until first time to reach node v.*

Definition 19 *The* **commute time** *between nodes u and v is the expected number of random walk steps required to go from u to v and back. That is*

$$C(u, v) = H(u, v) + H(v, u)$$

Notation: We denote by H_{max} the maximum expected hitting time between any pair of nodes in a network. That is

$$H_{max} = \max_{u,v} H(u, v)$$

Definition 20 *Given a network graph, the underlying* **electrical network** *is the network obtained by replacing vertices by nodes and edges by electrical resistances of a unit resistance. The* **effective resistance** *between any two nodes u and v, denoted by $R(u, v)$, can be defined as the voltage that develops between u and v when a unit current is maintained through them (i.e. enters one and leaves the other).*

4.3 Observations on General Networks

4.3.1 Two Packets to a Single Destination

Lemma 20 *Consider an arbitrary potential guided algorithm on any network, with two packets destined to the same node, each injected at an arbitrary time. Then each packet is delivered to its destination within $n + 1$ steps from its time of injection.*

Proof. Consider a path π of packet p in the network. Assume the packet was deflected at some time t, but met the other packet later on at some vertex (say at time $t' > t$). We can replace the interval $[t, t']$ in the path by the corresponding interval in the deflecting packet's path. Note that the two intervals are of the same length, but the deflection behavior is replaced by advancing one (since both packets are guided by the same potential function).

Let us repeat this procedure for all but the last deflection (which might have no posterior meeting), and denote this path π'.

Assume the last deflection (if any) occurs at time T. Then at time $T+1$ (after the deflection) p advances back to the deflecting node, or to one with lower potential. So $\Phi(\pi'(T+3)) < \Phi(\pi'(T+2)) \leq \Phi(\pi'(T))$.

Removing nodes $\pi'(T+1), \pi'(T+2)$ yields a (strictly) decreasing potential sequence, which therefore, can consist of at most n nodes. Hence, $|\pi| = |\pi'| \leq n+2$, so its length is no more than $n+1$ steps. □

4.3.2 The General Case of Two Packets

In this section we investigate the performance of a general potential guided algorithm in problems with only two packets. In this case, we can show the routing terminates in n steps, by charging deflections appropriately. The reader may wish to skip the proof, as it comes down to case analysis.

Throughout this proof, we say that the packet *advances* if it get its most preferred edge. Otherwise we say that the packet is *deflected.*

Observation 21 *Any potential guided algorithm on any network with only 2 packets satisfies the following:*

1. *The packets cannot meet at two successive time steps.*
2. *A packet is always deflected either to the node it came from, or to one with lower potential.*
3. *Every two steps, the potential of each packet either improves or remains unchanged.*
4. *If a packet has two successive advances after visiting some node, it cannot return to this node later on.*
5. *A packet can visit the same node only twice, in a time difference of two steps.*

Lemma 22 *Let T_i be the time packet p_i is delivered to its destination. Then, there exists a function $G : \{0, \ldots, T_i\} \mapsto 2^V$ such that $t \leq |G(t)| + 2$ for all $0 \leq t \leq T_i$.*

Proof. By induction on t. (Throughout this proof, the term potential at a node refers to the potential of packet p_i).

(i) For $t = 0$ define $G(0) = \{\}$, so it holds trivially.

(ii) Assume it holds for $t \geq 0$, and we shall see it holds also for $t+1$. Note that it suffices to show that $|G(t)| + 1 \leq |G(t+1)|$, or alternatively, $|G(t-1)| + 2 \leq |G(t+1)|$.

Define $G(t+1)$ inductively, as follows (recall that $S(p,t)$ denotes the location of packet p at time t):

1. If p_i advances at time t, define $G(t+1) = G(t) \cup \{S(p_i, t)\}$.
2. If p_i is deflected at time t for the first time in its route, define $G(t+1) = G(t-1)$.
3. If p_i is deflected at time t but was previously deflected at time $t' < t$, then we define $G(t+1)$ as follows. Assume that packet p_j was the deflecting packet at time t', i.e. $v = S(p_i, t') = S(p_j, t')$.

   Let w_i, w_j be the nodes from which the two packets p_i, p_j advance to the meeting point at time t, respectively (i.e. $w_i = S(p_i, t-1)$ and $w_j = S(p_j, t-1)$). Clearly, $w_i \neq w_j$, or we would have two successive meetings of the two packets. Let w be one of the w_i, w_j such that p_i is not deflected to it at time t (e.g. take the one with the higher potential w.r.t. p_i), as in figure 4.1.

   In the case that $S(p_i, t') = S(p_i, t'+2)$, we define another node, u. Consider the paths that p_i and p_j travel between their meeting at time t' and the next one at time t. Let u be the first node visited by p_j in the interval $[t', t]$, but not visited by p_i in the interval $[t'+2, t]$, say $u = S(p_j, t_1)$ (see figure 4.1). Such a node must exist before w_j by a counting argument. This node u cannot be v since v is visited by p_i at time $t'+2$. Note that p_i advances with no interference in the interval $[t'+2, t-1]$. Then at some time step, p_i had the option to advance to u, but chose to advance to some other node. Note that p_j is two steps ahead of p_i on their journey from v to u.

   Finally, we conclude the definition:

$$G(t+1) = \begin{cases} G(t-1) \cup \{v, w_j\} & \text{if } v \neq S(p_i, t'+2) \text{ and } w = w_j \\ G(t) \cup \{v\} & \text{if } v \neq S(p_i, t'+2) \text{ and } w = w_i \\ G(t-1) \cup \{u, w_j\} & \text{if } v = S(p_i, t'+2) \text{ and } w = w_j \\ G(t) \cup \{u\} & \text{if } v = S(p_i, t'+2) \text{ and } w = w_i \end{cases}$$

   Observe the following simple properties:

   (a) If v is added to $G(t)$ at time t, then p_i visits v only at time t'.
   Indeed, v is added only if it is not visited at time $t'+2$.

   (b) If u is added to $G(t)$ at time t, then p_i never visits u.

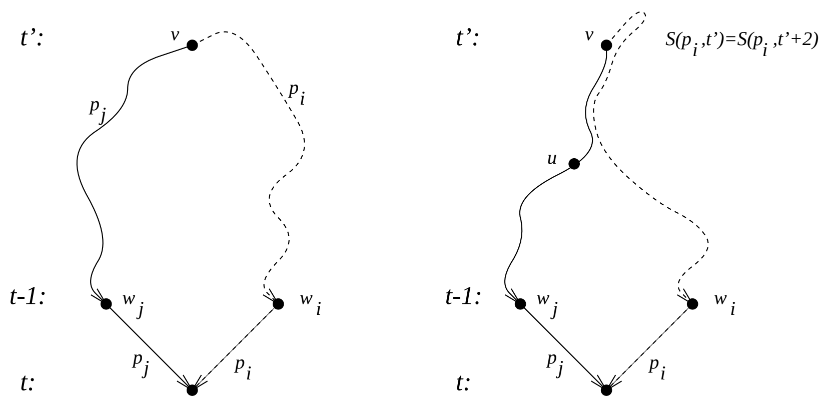

Figure 4.1: Nodes w_i, w_j (on the left) and u (on the right)

Indeed, p_i had the option to go to u, but advanced to another node, from which it advanced once again. It is therefore impossible for p_i to get to u later on.

(c) If w_j is added to $G(t)$ at time t, then p_i never visits w_j.
Indeed, w_j is added only if packet p_i is deflected to some other node, whose potential is lower.

(d) p_j visits w_j only at time $t-1$.
Indeed, p_j advances from w_j two successive steps.

Now that we have defined $G(t)$ properly, we can show the inductive phase. Note that in Case (2) above, $G(1), \dots, G(t-1)$ are all defined by Case (1), hence

$$G(t+1) = G(t-1) = \{S(p_i, 0), \dots, S(p_i, t-2)\}$$

But since $\Phi_{p_i}(S(p_i, 0)) > \cdots > \Phi_{p_i}(S(p_i, t-2))$, all these nodes are distinct. Therefore, $|G(t+1)| = t-1$.

All that remains to prove now is the induction hypothesis in cases (1) and (3). We will do that by proving that each element introduced to $G(\tau)$ in these cases, is indeed *new* to $G(\tau)$.

Let x be an arbitrary node which is introduced to $G(\tau)$ at some time τ. Assume to the contrary that x is not new in $G(\tau)$, but it is already in G, and was introduced at a previous time $t < \tau$. Case (3) introduces 3 types of elements: v, u and w_j. So together with Case (1), we have a total of 4 ways to introduce a new element to G. This sums up to 16 possibilities to introduce x to G twice, at times t and at time τ, and we will see that each of them is impossible.

Suppose that x is added to $G(\tau)$ according to Case (1), so p_i advances from $S(p_i, \tau)$ at time τ. In particular, p_i visited x at time τ. We consider all 4 types of elements in which x could have been added to G previously, at time $t < \tau$.

- Assume that x was previously added (at time $t < \tau$) according to Case (1). Then p_i advances twice from the same node, which can happen (Observation 21) only if $t = \tau - 2$ and p_i was deflected at time $\tau - 1$. But, in this deflection the node x which was added at time $\tau - 2$ is removed from G, so it is new to $G(\tau)$. Contradiction.
- Assume that x was previously added (at time $t < \tau$) according to Case (3) as v. By Property (3a), this node x is visited by p_i only at the corresponding time $t' < t < \tau$. It clearly cannot be visited by p_i at time τ. Contradiction.
- Assume that x was previously added (at time $t < \tau$) according to Case (3) as u. By Property (3b), this node x is never visited by p_i. It clearly cannot be visited by p_i at time τ. Contradiction.
- Assume that x was previously added (at time $t < \tau$) according to Case (3) as w_j. By Property (3c), this node x is never visited by p_i. It clearly cannot be visited by p_i at time τ. Contradiction.

Suppose that x is added to $G(\tau)$ according to Case (3) as $v_\tau = S(p_i, \tau')$. Then, p_i was deflected from x at time τ', and by Property (3a), this is the only visit of p_i to x. We consider all 4 types of elements in which x could have been added to G previously, at time $t < \tau$.

- Assume that x was previously added (at time $t < \tau$) according to Case (1). Then p_i advances from x at time $t' \neq \tau'$. But the only visit of p_i to x is at time τ'. Contradiction.
- Assume that x was previously added (at time $t < \tau$) according to Case (3) as v. Then p_i was deflected from x at time $t' \neq \tau'$. But the only visit of p_i to x is at time τ'. Contradiction.
- Assume that x was previously added (at time $t < \tau$) according to Case (3) as u. By Property (3b), this node x is never visited by p_i. It clearly cannot be visited by p_i at time τ'. Contradiction.
- Assume that x was previously added (at time $t < \tau$) according to Case (3) as w_j. By Property (3c), this node x is never visited by p_i. It clearly cannot be visited by p_i at time τ'. Contradiction.

Suppose that x is added to $G(\tau)$ according to Case (3) as u_τ. Then by Property (3b), x is never visited by p_i. We consider all 4 types of elements in which x could have been added to G previously, at time $t < \tau$.

- Assume that x was previously added (at time $t < \tau$) according to Case (1). Then p_i advances from x at time $t' \neq \tau'$. But x is never visited by p_i. Contradiction.
- Assume that x was previously added (at time $t < \tau$) according to Case (3) as v. Then p_i was deflected from x at time $t' \neq \tau'$. But x is never visited by p_i. Contradiction.
- Assume that x was previously added (at time $t < \tau$) according to Case (3) also as u.

  By the definition of u, p_j visited x twice. By Observation 21, the second visit must be two steps after the first one, i.e. $\tau_1 = t_1 + 2$. From time t' up to the first visit at node u at time t_1, packet p_j is 2 steps ahead of p_i. Thus, p_j cannot meet p_i at time t_1 (at node u) nor in the following time step. Contradiction.
- Assume that x was previously added (at time $t < \tau$) according to Case (3) as w_j. By Property (3d), p_j visits x only at time $t-1$. So by the definition of u, $t = \tau$ is the same deflection. Contradiction.

Suppose that x is added to $G(\tau)$ according to Case (3) as $w_{j,\tau} = S(p_j, \tau - 1)$. Then by Property (3c), x is never visited by p_i. We consider all 4 types of elements in which x could have been added to G previously, at time $t < \tau$.

- Assume that x was previously added (at time $t < \tau$) according to Case (1). Then p_i advances from x at time $t' \neq \tau'$. But x is never visited by p_i. Contradiction.
- Assume that x was previously added (at time $t < \tau$) according to Case (1). Then p_i advances from x at time $t' \neq \tau'$. But x is never visited by p_i. Contradiction.
- Assume that x was previously added (at time $t < \tau$) according to Case (3) as u. By Property (3d), x is visited by p_j only at time $\tau - 1$. But by the definition of u, this implies that $t = \tau$ is the same deflection. Contradiction.
- Assume that x was previously added (at time $t < \tau$) according to Case (3) as w_j. By Property (3d) for t, p_j visits x only at time $t-1$. By Property (3d) for τ, p_j visits x only at time $\tau - 1$. Then $t = \tau$ is the same deflection. Contradiction.

□

Corollary 23 *Let T_i be the time packet p_i is delivered to its destination. Then there exists a function $H : \{0, \ldots, T_i\} \mapsto 2^V$ such that $t \leq |G(t)| + 1$ for all $0 \leq t \leq T_i$.*

Proof. Follow the proof of Lemma 22, and change Case (2) in the definition of $G(t+1)$ to $G(t+1) = G(t-1) \cup \{w\}$, where w is defined as in Case (3). The rest of the proof still holds. □

Corollary 24 *Any potential guided algorithm routes any problem with two packets, in $(n-1)+1 = n$ steps.*

4.4 Livelock Free Algorithms

Recall that if for every packet, the potential at a node is higher than the average potential at its neighbors (except for the destination node), then the potential guided algorithm is guaranteed to terminate.

In this section we show that although this property avoids livelocks, it does not ensure $t_p \leq d_p + 2(k-1)$. We demonstrate such a potential guided algorithm with a problem consisting of 4 packets.

Observation 25 *There are certain networks of n nodes ($n \geq 13$), satisfying the property that the potential at each node is higher than the average potential at its neighbors, such that a potential guided algorithm routing a permutation problem of 4 packets ($k = 4$) achieves $t_p = 2n - 7$, for some packet p.*

Proof. Consider a pendulum network with $n-2$ nodes on a cycle (say $v_1, v_2, \ldots, v_{n-2}$), and 2 nodes (say v_{n-1}, v_{n-2}) connected to the cycle one after the other, as in figure 4.2.

Denote the four packets as p_1, p_2, p_3, p_4. Let p_1 originate at v_{n-7}, and be destined to v_n (see figure 4.2). Assume p_1's potential function is the following:

$$\Phi_1(v_i) = \begin{cases} 11 + \sqrt{i} & \text{if } i \leq n-4 \\ 12 & \text{if } i = n-3 \\ 11 & \text{if } i = n-2 \\ 6 & \text{if } i = n-1 \\ 0 & \text{if } i = n \end{cases}$$

Note that the potential Φ_{p_1} at each node is higher than the average potential at its neighbors.

Let p_2, p_3 and p_4 originate at v_5, v_3 and v_1 and be destined to v_{n-1}, v_{n-2} and v_{n-3}, respectively. Assume the potential functions for p_2, p_3 and p_4 decreases along the cycle in the direction that i grows, until v_{n-2}. So each packet will go to its destination along the longer side of the cycle. The property of higher potential at a node than the average of its neighbors is easy to satisfy by using an arbitrary convex function (like square root). Finally, we make sure that p_2, p_3 and p_4 have "priority" over p_1 by stronger potential differences (say multiples of $1000n$).

Packets p_2, p_3, p_4 travel along the cycle one after the other with 2 nodes difference between each pair of successive packets, and they never meet each other. Packet p_1

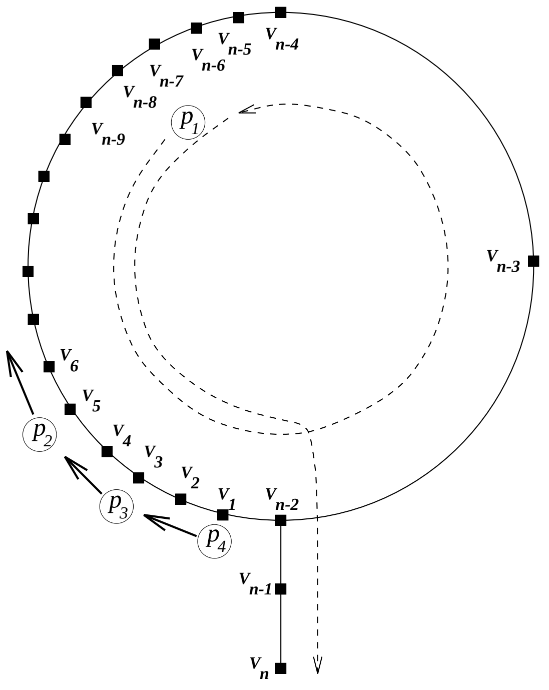

Figure 4.2: Four packets on an upside down pendulum

travels in the opposite direction, until meeting p_2 at v_{n-2} after $n-7$ steps. Here, they both want to advance to v_{n-1}, but p_2 has higher "priority", so p_2 advances, and p_1 is deflected to v_{n-3}, where it meets p_3 (after $n-6$ steps). Since p_3 has higher "priority", it advances to v_1, and p_1 is deflected to v_{n-4}, where it meets p_4 (after $n-5$ steps). Since p_4 has higher "priority" it advances to v_{n-3}, and p_1 is deflected to v_{n-5}.

After the 3 successive deflections (see the dashed arrow in figure 4.2), p_1 will choose to go once again down the v_i's, namely $v_{n-5}, v_{n-6}, \ldots, v_2, v_1, v_{n-2}, v_{n-1}, v_n$. Hence, the total time required for p_1 to reach its destination is $(n-7)+(n-2)+2 = 2n-7$. $\square$

4.5 Tree Networks

In this section we investigate the performance of potential guided algorithms on tree networks. The definite structure of tree graphs (no cycles) enable us to analyze some characteristics of potential guided algorithms.

Observation 26 *Any potential function on a tree network is monotonic w.r.t. the distance to destination, i.e. the potential decreases iff one advances towards its destination.*

Proof. Assume to the contrary that for some packet, the node u is closer to destination than v, but has higher potential. Then v must have a neighbor v_1 with even lower potential (by the potential function definition). If v_1 is not the destination node, it must also have a neighbor v_2 with even lower potential, and so on. As long as the destination is not reached, we can find a neighbor with lower destination. However, each v_i has a lower potential then the previous one, and definitely lower than u's potential. So $\forall i,\ v_i \neq u$ and the process never reaches u. Every path from v to the destination must go through u. Thus, the destination node is never reached. Since the graph size is finite, some node must be repeated, contradicting the decrease in potential. □

Corollary 27 *In any potential guided algorithm on any tree network, every packet has exactly one preferred edge, and all other outgoing edges are deflections.*

Observation 28 *There are potential guided algorithms on tree networks which are not maximum advance.*

Proof. We show an example of applying the "minimum sum of potentials" rule, where the number of packets which improve their potential is not maximal (see figure 4.3). Recall that advancing a packet on a tree network is equivalent to improving its potential (by Corollary 27). Hence, this assignment is not maximum advance. □

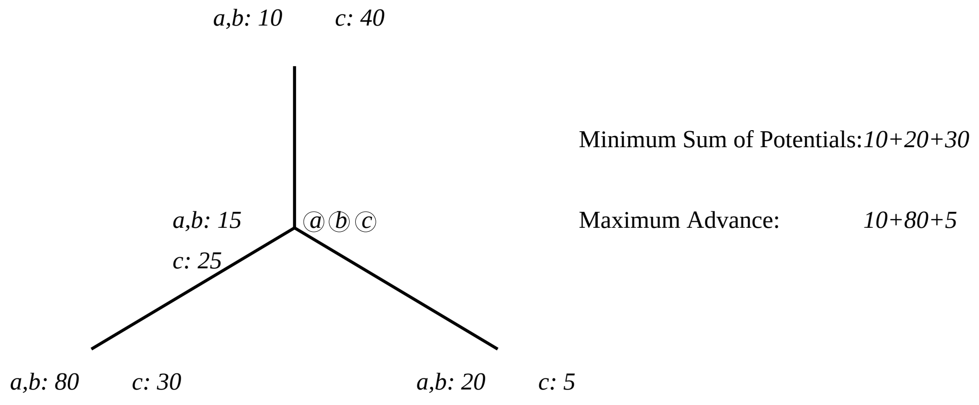

Figure 4.3: A potential guided algorithm on tree network which is not maximum advance.

Observation 29 *Any potential guided algorithm on any tree network is stable.*

Proof. Let A be the optimal assignment of outgoing edges in some node, i.e. it achieves the minimum sum of potentials. Assume to the contrary that there is a possibility to change the assignment of some of the packets in A, such that all those packets strictly gain from the change, meaning that they get an edge that leads them closer to their destination. By Corollary 27, advancing in this network is equivalent to improving the potential, so the potential of each of these packets can be improved. Clearly, the "sum of potentials" can be improved by this change, contradicting the optimality of assignment A. $\square$

Corollary 30 *Any potential guided algorithm evacuates a tree network, without entering a livelock.*

Proof. Follows immediately from Corollary 18, regarding Minimum Advance algorithms on tree networks. $\square$

4.6 Expected Hitting Time Potential

In this section we consider a specific potential function, which is the expected hitting time (of a random walk) form the packet's current location to its destination. We refer to this algorithm as the *expected hitting time routing algorithm*, and investigate its performance on general networks as well as on specific networks, particularly, trees and cycles.

4.6.1 General Case Analysis

The hitting time of random walks on general graphs was analyzed by Aleliunas et al. [2]. One of their results is a bound for the expected hitting time in general graphs, $H_{max} \leq 2mD$, where m is the number of edges in the graph, and D is its diameter.

An improved bound for H_{max} was presented by Chandra et al. [10]. Their work links the commute time between two nodes to the effective resistance between them (in the underlying electrical network), $C(u, v) = 2m \cdot R(u, v)$. Since the commute time bounds the expected hitting time, this yields that $H_{max} \leq 2m \cdot R_{max}$.

On a general graph of n vertices, these results can be bounded as a function of n alone. The diameter D (and also the maximum electrical resistance R_{max}) is at most n, and $m < \frac{1}{2}n^2$, yielding the upper bound $H_{max} \leq n^3$.

Lemma 31 *Consider a routing problem on any network with k packets. Then the expected hitting time routing algorithm evacuates at least half of the packets in the network, every* $2H_{max}$ *steps.*

Proof. By the definition of the expected hitting time potential, the potential at node is exactly 1 more than the average potential at its neighbors. Hence, a random assignment of a packet to an outgoing edge is expected to reduce the potential by 1. A random assignment of l packets in a node is expected to reduce the sum of potentials by l. Therefore, there must exist some assignment of the l packets which reduces the sum of potentials by at least l. The expected hitting time routing algorithm chooses the assignment with lowest sum of potentials, and thus reduces the sum of potentials by at least l.

Consider the sum of potentials of all packets in the network. Initially, it is bounded by $k \cdot H_{max}$ since the potential of each packet is no more than H_{max}. At each step, the overall sum of potentials is reduced at least by the current total number of packets in the network.

Assume to the contrary, that after $2H_{max}$ steps, the number of packets in the network is still more than $\frac{1}{2}k$. At each of the $2H_{max}$ steps, the sum of potentials is reduced by at least $\frac{1}{2}k$. So the total sum of potentials after the H_{max} steps will be at most $kH_{max} - \frac{1}{2}k \cdot 2H_{max} = 0$, contradicting the existence of packets in the network. □

Corollary 32 *The expected hitting time routing algorithm evacuates any problem of k packets on any network within $2H_{max} \log k$ steps.*

4.6.2 Tree Networks

Tree graphs have a simple layout, which is easier to analyze. We show that the expected hitting time is maximum advance, and thus delivers any packet to its destination within $d_p + 2(k-1)$ steps.

Any two vertices on a tree graph are connected by only one simple path. Therefore, a packet entering a node has exactly one advancing direction. That is, one neighbor is closer to destination, and all other neighbors are further away from the packet's destination.

Observation 33 (triangle equality) *If v is located on a simple path from u to w, then*

$$H(u,v) + H(v,w) = H(u,w)$$

Proof. Trivially, every path (walk) from u to w must go through v. The proof follows by linearity of expectation. □

Corollary 34 *In an expected hitting time routing algorithm on a tree network, every packet prefers only the edge on which it advances towards its destination.*

Remark: Corollary 34 is a private case of Corollary 27 with expected hitting time as the potential function.

Lemma 35 *The expected hitting time routing algorithm is maximum advance on tree networks.*

Proof. Assume to the contrary that the expected hitting time routing algorithm is not maximum advance on trees. Then there is a node u in a tree network T, where the assignment with the minimum sum of potentials, A_{eht} , advances fewer packets than the maximum advance assignment, A_{max}. Assume that A_{eht} advances $s \geq 0$ packets, and A_{max} advances $r > s$ packets.

Note that the number of outgoing edges assigned with advancing packets is r by assignment A_{max}, and only $s < r$ by A_{eht}. By the pigeon-hole principle, there must be some edge $e_1 = (u, w_1)$ which is assigned with an advancing packet (say p) in A_{max}, but it is not assigned with an advancing packet in A_{eht}. Since A_{eht} does not assign p to e_1 (it would be an advancing assignment), then p must be assigned to some other edge $e_2 = (u, w_2)$, which is a deflection. See figure 4.4 for illustration of A_{max} vs. A_{eht}.

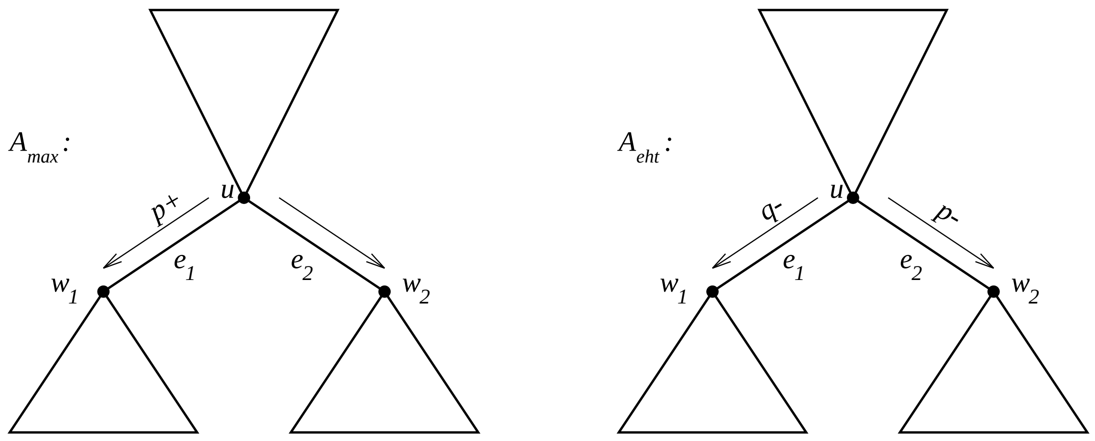

Figure 4.4: A_{max} (on the left) vs. A_{eht} (on the right).

A_{eht} must assign some packet to this e_1, or otherwise A_{eht} could be improved by assigning p to advance on e_1 instead of being deflected on e_2. This would contradict the optimality of A_{eht}. Let q be the packet assigned by A_{eht} to e_1. It is a deflection since e_1 is not one of the r edges which are assigned with an advancing packet by A_{eht}.

Swap the assignments of p and q in A_{eht} and denote this new assignment A'. We will see that A' has better "sum of potentials" then A_{eht}, contradicting the optimality of A_{eht}. If q advances on e_2, then A' is clearly better than A_{eht} (the swap improves both p and q). So we assume that q is deflected in A' (on e_2).

$$\begin{aligned}\Phi_u(A_{eht}) &= \Phi_p(w_2) + \Phi_q(w_1) + \sum_{p' \neq p,q} \Phi_{p'}(A_{eht}(p')) \\ \Phi_u(A') &= \Phi_p(w_1) + \Phi_q(w_2) + \sum_{p' \neq p,q} \Phi_{p'}(A_{eht}(p'))\end{aligned}$$

By Observation 33, we get that

$$\begin{aligned}
\Phi_p(w_2) &= H(w_2, u) + H(u, w_1) + \Phi_p(w_1) \\
\Phi_q(w_1) &= H(w_1, u) + \Phi_q(u) \\
\Phi_q(w_2) &= H(w_2, u) + \Phi_q(u)
\end{aligned}$$

and we're done:

$$\Phi_u(A_{eht}) - \Phi_u(A') \;=\; H(u, w_1) + H(w_1, u) \;>\; 0$$

□

Corollary 36 *The expected hitting time routing algorithm on tree networks delivers any packet to its destination within $d_p + 2(k-1)$ steps.*

Proof. Delivery time of $d_p + 2(k-1)$ is guaranteed by Feige [11], in any maximum advance hot-potato routing algorithm on tree networks, and, in particular, for the expected hitting time routing algorithm. □

4.6.3 Cycle Networks

In this section we analyze the performance of the expected hitting time routing algorithm on cycle networks. The simple structure of cycles enables to compute the expected hitting time between any two nodes in the cycle. Directions of advance and deflection are easy to analyze, due to the simple layout of the cycle network.

We link the difference in the packet's potential to these directions, (i.e. potential decreases on advances and increases on deflections). This is used to show that the expected hitting time routing algorithm on a cycle network is maximum advance with priority to close destinations, and thus, delivers each packet to its destination in $t_p \leq d_p + 2(k-1)$ steps.

Commute times on the cycle are easy to compute using the rule devised by Chandra et al. [10]. Applying symmetry argument yields that $H(u, v) = \frac{1}{2}C(u, v) = d(n-d)$ for any two nodes u and v whose distance on the cycle is d.

Observation 37 *The expected hitting time potential function on cycle networks is monotonic w.r.t. distance from destination, i.e. the potential decreases iff one advances towards its destination.*

Proof. Consider a cycle of n nodes. Then, for any two nodes u and v on the cycle, $d(u, v) \leq \frac{n}{2}$. The function $f(d) = d(n-d)$ is (strictly) monotonic on the interval $[0, \frac{d}{2}]$, so the packet's potential and the distance to destination increase and decrease together. □

Corollary 38 *The expected hitting time routing algorithm is maximum advance on cycle networks.*

Proof. The number of packets in each node is at most 2, since the degree of a node is 2. A single packet will always advance towards its destination, since an advancing edge always has smaller potential than a deflecting one.

Consider a node with two packets to assign. Let A_{eht} be the assignment of the expected hitting time routing algorithm in this case, and let p be a deflected packet in A_{eht}. The other packet, p', strictly prefers the same edge as p, or otherwise it would be possible to improve the assignment by swapping the packets (note that the assignment is improved even when p is at an equilibrium node where $d = \frac{n}{2}$). Hence, the two packets contest on the same edge. But only one packet can be assigned to this preferred edge, so A_{eht} is maximum advance. □

Corollary 39 *The expected hitting time routing algorithm on a cycle prefers close destinations.*

Proof. At any time no more than two packets can enter a node in the network, since the degree of each node is 2. Let u be a node with 2 packets p_1 and p_2, who both prefer the same outgoing edge, $e = (u, v)$. Let d_1, d_2 be the distances from u to p_1's and p_2's destinations, respectively, and suppose that $d_1 < d_2$. We will show that p_1 has priority over p_2.

Let A_1 be the assignment which advances p_1 on e, and deflects p_2. Let A_2 be the assignment which advances p_2 on e, and deflects p_1. So our goal is to show that $\Phi(A_2) > \Phi(A_1)$.

When p_i advances, its distance from destination is shortened by 1, and it's potential will be $\Phi_{p_i}(d_i - 1)$. However, when p_i is deflected, its distance from destination either increases by 1 or remains unchanged (in the case where $d_i = \frac{n-1}{2}$). The formula for the expected hitting time on the cycle, $H(u, v) = d(n - d)$, holds also in the latter case because it is symmetric with d and $n - d$. So in any case, the potential of p_i after the deflection will be $\Phi_{p_i}(d_i + 1)$.

$$\begin{aligned}
\Phi(A_1) &= \Phi_{p_1}(d_1 - 1) + \Phi_{p_2}(d_2 + 1) \\
&= (d_1 - 1)(n - d_1 + 1) + (d_2 + 1)(n - d_2 - 1) \\
\Phi(A_2) &= \Phi_{p_1}(d_1 + 1) + \Phi_{p_2}(d_2 - 1) \\
&= (d_1 + 1)(n - d_1 - 1) + (d_2 - 1)(n - d_2 + 1)
\end{aligned}$$

and we're done:

$$\Phi(A_2) - \Phi(A_1) = 4(d_2 - d_1) > 0$$

□

Lemma 40 *The expected hitting time routing algorithm on cycle networks routes a packet p to its destination in $t_p \leq d_p + 2(k - 1)$.*

Proof. We first claim that a packet can be deflected no more than $k-1$ times, using the "chain of deflections" argument. If a packet p is deflected, then there is another packet which advances in p's preferred direction and so on, creating a chain of deflections along a simple path.

Since the algorithm gives priority to close destinations, the destinations of packets in the chain cannot get further away (only stay in place or get closer). In particular, all destinations are not beyond $\frac{1}{2}n$ far from the beginning of the chain. Therefore, the chain length is bounded by $\frac{n}{2}$, ensuring uniqueness of the last packet in the chain. Hence, each packet can suffer no more than $k-1$ deflections.

Consider the route traveled by a packet p for $d_p + 2(k-1)$ steps. At most $k-1$ of these steps can be deflections. Each deflection increases the distance to destination by at most 1. The remaining steps are advances, which reduce the distance to destination by 1. Therefore, the distance of p from its destination after traveling $d_p + 2(k-1)$ steps, is bounded by:

$$d_p + \underbrace{(k-1)}_{deflections} \cdot 1 - \underbrace{(d_p + 2(k-1) - (k-1))}_{advances} \cdot 1 = 0$$

so p must have reached its destination within $t_p \leq d_p + 2(k-1)$ steps. $\square$

4.6.4 Mesh Networks

The expected hitting time is guaranteed to evacuate any network, and in particular, an $n \times n$ mesh within $2H_{max} \log k$ steps. Chandra et al. [10] show that on mesh graphs $R_{max} = O(\log n)$, ensuring evacuation within $O(n^2 \log n \log k)$ steps.

Figure 4.5 demonstrates the potential (expected hitting time) of a packet p destined to node $(3,5)$, at each node of the 11×11 mesh. Arrows denote the preferred outgoing edges from each node to this destination, $(3,5)$. A single packet destined to $(3,5)$ will travel along these edges with no interference, until reaching its destination. However, traveling along these edges is not always the shortest possible. For example, the preferred path of a packet generated at $(11,11)$ is definitely not the shortest possible to $(3,5)$.

A packet's route in a expected hitting time routing algorithm on a two dimensional mesh network is not necessarily a shortest path, even if the packet is traveling alone in the network. Although the expected hitting time routing algorithm achieves $t_p \leq d_p + 2(k-1)$ on trees and cycles, it does not achieve this bound on other simple networks such as meshes. We also see that the expected hitting time routing algorithm is maximum advance on tree and cycle networks, but is not even minimum advance on mesh networks.

4.6.5 Open Problems

Several interesting open questions remain. One major goal is to further investigate the behavior of the expected hitting time routing algorithm on general networks. Our

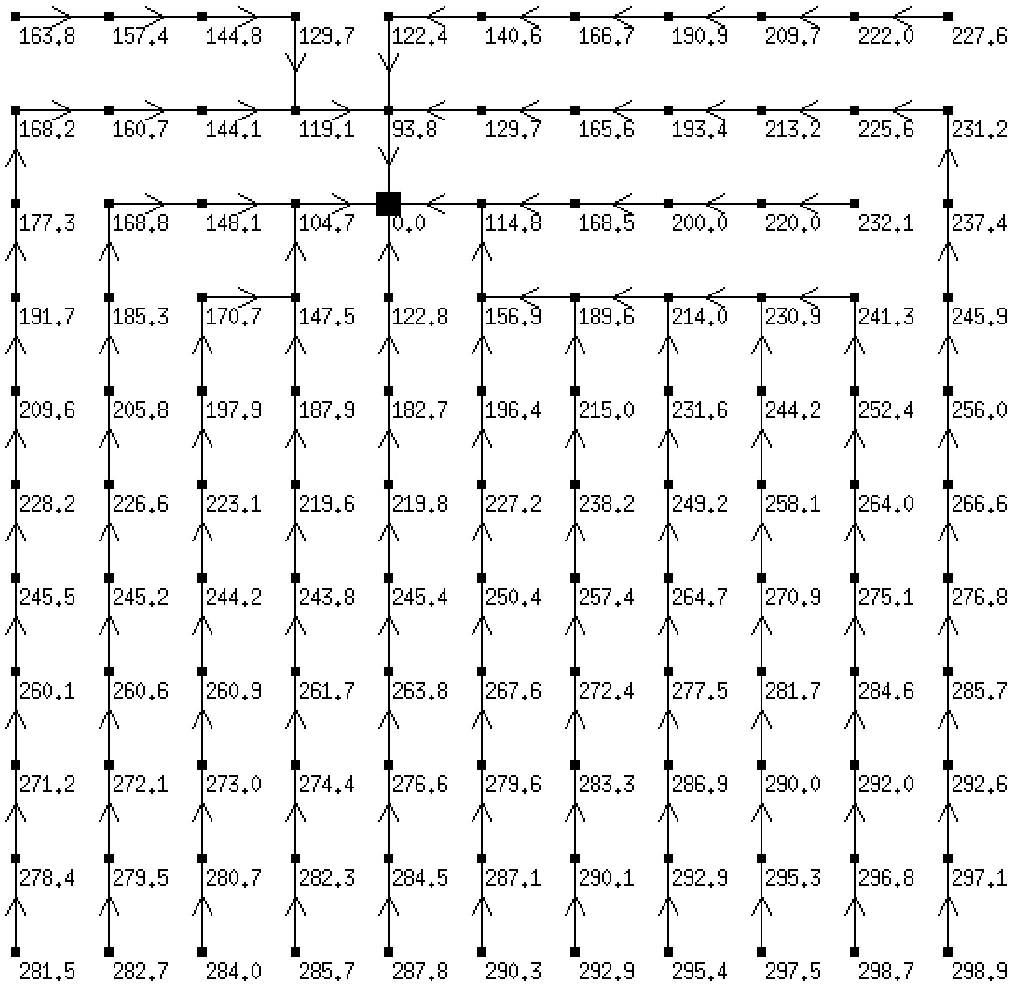

Figure 4.5: Expected hitting times to destination $(3, 5)$ on the 11×11 mesh

lower and upper bounds (e.g. on meshes) leave a gap which contains some interesting bounds, such as $O(D + k)$ and $(n + k)$. The cases of one and two packets suggest that the general case upper bound $(2n^3 \log k)$ can be improved.

Another interesting problem is the behavior of a single packet on specific networks. It is not well understood why a single packet does not follow a shortest path, and the question of bounding the path traveled by a single packet in a network (in terms of d_p or D) remains open.

Finally, it would be interesting to devise other potential functions which can be shown to perform well on specific or general networks.

Bibliography

[1] A.S. Acampora, S.I.A. Shah. "Multihop lightwave networks: a comparison of store-and-forward and hot-potato routing". *Proc. IEEE INFOCOM,* pp. 10-19. IEEE Computer Societ Press, 1991.

[2] R. Aleliunas, R. M. Karp, R. J. Lipton, L. Lovasz, C. Rackoff. "Random walks, universal traversal sequences and the complexity of maze problems". *20th Annual Symposium on Foundations of Computer Science,* pp. 218-223, San Juan, Puerto Rico, October 1979.

[3] P. Baran. "On distributed communication networks". *IEEE Transactions on Communications,* pp. 1-9, 1964.

[4] A. Bar-Noy, P. Raghavan, B. Schieber, H. Tamaki. "Fast deflection routing for packets and worms". *Proc. 12th ACM Symp. on Principles Of Distributed Computing,* 1993.

[5] I. Ben-Aroya, A. Schuster. "Greedy Hot-Potato Routing on the Mesh". *ESA: Annual European Symposium on Algorithms,*, 1994.

[6] A. Ben-Dor, S. Halevi, A. Schuster. "Potential Function Analysis of Greedy Hot-Potato Routing". *Proc. of 13th ACM Symp. on Priciples of Distributed Computation,* pp. 225-234, 1994.

[7] A. Borodin, J.E. Hopcroft. "Routing, merging and sorting on parallel models of computation". *JCSS,* 30:130-145, 1985.

[8] A. Borodin, Y. Rabani, B. Schieber. "Deterministic many-to-many hot-potato routing". *manuscript,* 1994.

[9] J.T. Brassil, R.L. Cruz. "Bounds on Maximum Delay in Networks with Deflection Routing". *29th Annual Allerton Conference on Communication, Control, and computing,* 571-580, 1991.

[10] A. K. Chandra, P. Raghavan, W. L. Ruzzo, R. Smolensky, P. Tiwari. "The electrical resistance of a graph captures its commute and cover times". *Proceedings of the 21st Annual ACM Symposium on Theory of Computing,* pp. 574-586, Seattle, WA, May 1989.

[11] U. Feige. "Observations on Hot Potato Routing". *Proc. of third Israel Symposium on the Theory of Computing and Systems,* 30-39, 1995.

[12] U. Feige, P. Raghavan. "Exact analysis of hot-potato routing". *Proc. IEEE symposium on Foundations of Computer Science,* pp. 553-562, 1992.

[13] A.G. Greenberg, J. Goodman. "Sharp approximate models of adaptive routing in mesh networks." In O.J. Boxma, J.W. Cohen, H.C. Tijms, editors, *Teletraffic Analysis and Computer Performance Evaluation,* pp. 255-270. Elsevier, Amsterdam, 1986. Revised 1988.

[14] D. Greene, M. Parnas, F. Yao. "Multi-Index Hashing for Information Retrieval". *Proceedings of the 35th Annual IEEE Symposium on Foundations of Computer Science,* 1994.

[15] B. Hajek. "Bounds on evacuation time for deflection routing". *Distributed Computing,* 5:1-6, 1991.

[16] W.D. Hillis. *The Connection Machine.* MIT Press, 1985.

[17] C. Kaklamanis, D. Krizanc, S. Rao. "Hot-Potato routing on processor arrays". *Proc. ACM Symp. Parallel Algorithms and Architecture,* pp. 273-282, 1993.

[18] M. Kaufmann, H. Lauer, H. Schroder. "Fast deterministic hot-potato routing on processor arrays". *ISAAC: 5th International Symposium on Algorithms and Computation,* 1994.

[19] A. Lempel, J. Ziv. "Compression of two-dimensional data". *IEEE Trans. Inform. Theory,* Vol. IT-32, no. 1, pp. 2-8, 1986.

[20] N. Linial, O. Sasson. "Non-Expansive Hashing". *Proceedings of the 28th Annual Symposium on Theory of Computing,* 509-518, 1996

[21] N.F. Maxemchuk. "Comparison of deflection and store and forward techniques in the manhattan street and shuffle exchange networks." *IEEE INFOCOM,* pp. 800-8009, 1989.

[22] M. Minsky, S. Papert. "Perceptrons". *MIT Press, Cambridge, Massachusetts,* pp. 222-225, 1969.

[23] I. Newman, A. Schuster. "Hot-Potato Algorithms for Permutation Routing." *IEEE Transactions on Parallel and Distributed Systems,* Vol. 6, No. 11, pp. 1168-1176, 1995.

[24] M. Parnas. “Robust Algorithms and Data structures for information retrieval”. *Ph.D. dissertation, Hebrew University, Jerusalem, Israel,* 1994.

[25] R. Prager. “An algorithm for routing in hypercube networks”. *Master's Thesis, University of Toronto,* September 1986.

[26] B. Smith. “Architecture and applications of the HEP multiprocessor computer system.” *Proc. (SPIE) Real Time Signal PRocessing IV,* pp. 241-248, 1981.

[27] T. Szymanski. “An analysis of hot-potato routing in a fiber optic packet switched hypercube.” *Proc. IEEE INFOCOM,* pp. 918-926, 1990.

[28] Z. Zhang, A.S. Acampora. “Performance analysis and multihop lightwave with hot-potato routing and distance age priorities”. *Proc. IEEE INFOCOM,* pp. 1012-1021, IEEE COmputer Society Press, 1991.